AFRL-RI-RS-TR-2008-108
Final Technical Report
April 2008

DYNAMIC SITUATION ASSESSMENT AND PREDICTION (DSAP) INTEGRATION AND EXPERIMENTATION FOR C2 R&D

RAM Laboratories, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2008-108 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/	/s/
DAWN A. TREVISANI Work Unit Manager	JAMES W. CUSACK Chief, Information Systems Division Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
APR 2008	Final	May 07 – Feb 08

4. TITLE AND SUBTITLE

DYNAMIC SITUATION ASSESSMENT AND PREDICTION (DSAP) INTEGRATION AND EXPERIMENTATION FOR C2 R&D

5a. CONTRACT NUMBER
FA8750-07-C-0091

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Robert McGraw

5d. PROJECT NUMBER
459S

5e. TASK NUMBER
N7

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
RAM Laboratories, Inc.
10525 Vista Sorrento Parkway, Suite 220
San Diego CA 92121-2747

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2008-108

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-1703

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The DSAP concept grew out of John R. Surdu's Simulation in Operations research project and prototype system (OpSim). There are two basic functions of the DSAP concept: (1) Dynamic Situation Awareness, and (2) Prediction. The overall concept involves the use of embedded simulation in an operational environment to support decision makers in the planning process. Providing this capability allows decision makers to use simulation to assist in planning operations, monitor current operations, determine deviations from a plan, and predict and determine outcomes of a given plan. The resulting system allows military Commanders to utilize timely battlefield information to make accurate decisions based on the effects of their plans and the current operational picture while providing an underlying capability to allow the Commander to dynamically modify an existing plan "on the fly" to address emerging information detected by sensors or provided by gathered intelligence.

15. SUBJECT TERMS
Dynamic Situation Assessment, DSAP, OpSim, Dynamic Operational Planning, Operational Simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dawn A. Trevisani
U	U	U	UL	50	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

List of Figures

1.0 Introduction

In order to develop decision science technologies for future Air Operations Centers (AOCs), the Air Force Research Laboratory (AFRL) has defined a program to conduct and sponsor research and development of revolutionary decision-support concepts. A goal of this program is to apply advanced information technologies to promote tactics, techniques and procedures that enable situational awareness and predictive capabilities for future AOCs. One effort that addresses this goal involves the development of a Dynamic Situation Assessment and Prediction (DSAP) Framework. To address predictive analysis of plans in the area of decision support, RAM Laboratories has developed a DSAP Framework and its underlying Multiple Replication Framework (MRF) for AFRL/IFSB. A subsequent DSAP implementation provided Operational Situational Awareness for DSAP through the use of ideal real-time simulation and calibrated real-time simulation. Both the predictive and dynamic situational awareness implementations of DSAP leveraged the state of the art in advanced information management techniques in the fields of simulation, distributed computing, and information management to provide the underlying functionality to evaluate Commander's plans and their alternatives using predictive simulation while calibrating with the real-time Command Control Communications and Computers Intelligence (C4I) picture. This effort performed additional research into readying DSAP for use in experimentation and exercise support. This Final Report details efforts of RAM Laboratories in performing this research.

1.1 Significance to the Air Force

The DSAP concept grew out of John R. Surdu's Simulation in Operations research project and prototype system (OpSim). There are two basic functions of the DSAP concept: (1) Dynamic Situation Awareness, and (2) Prediction. The overall concept involves the use of embedded simulation in an operational environment to support decision-makers in the planning process. Providing this capability allows decision makers to use simulation to assist in planning operations, monitor current operations, determine deviations from a plan, and predict and determine outcomes of a given plan. The resulting system allows military Commanders to utilize timely battlefield information to make accurate decisions based on the effects of their plans and the current operational picture while providing an underlying capability to allow the Commander to dynamically modify an existing plan "on the fly" to address emerging information detected by sensors or provided by gathered intelligence.

The DSAP Framework builds on advanced information technologies to provide a predictive analysis of existing plans and alternatives within the context of the real-time operational picture. There are five main components of the DSAP Framework: (1) the Multiple Replication Framework (MRF) that is used for evaluating plans and alternative Courses of Action (COAs) against campaign objectives on available processors, (2) the Simulation Framework that allows real-time simulation and faster-than-real-time simulations to be developed in a manner that calibrates with real-time data and supports rollback/rollforward capabilities that can be used to track the real-time operational picture, (3) an Optimization Framework that supports plan generation through the use of simulation-in-the-loop, (4) the simulation component which simulates plans/COAs and their alternatives in both a real-time and predictive fashion (through the use of faster-than-real-time simulation, and (5) real-time databases and data feeds. This

framework provides the capability to dynamically assess situations, dynamically predict the outcomes of plans, and will be used to enhance the plan generation process.

RAM Laboratories, with the Air Force Research Laboratory previously developed a prototype DSAP Framework that provides a predictive capability and dynamic situational awareness capability. The predictive capability was provided through the use of faster-than-real-time predictive simulation using JSAF, and real-time data for calibration from the Theater Battle Management Core System (TBMCS) Air Operations Database (AODB) and Modernized Integrated Database (MIDB). The dynamic situational awareness capability was provided by implementing and integrating real-time simulation components with additional real-time databases and data feeds and the existing predictive analysis capability. This effort enhanced DSAP for eventual use in an experiment or exercise by streamlining the predictive and dynamic situational awareness control in order to run both modes simultaneously. The effort also worked to better segment functionality of MRF components in order to provide an improved plug-and-play capability and promote re-use of 3rd party evaluation, simulation, and planning applications/modules.

In addition to the above capabilities, this effort also installed the DSAP Framework in a laboratory setting to provide a demonstratable capability that can be transitioned to the operational domain and used as a building block for implementing additional decision-support technologies.

The subsequent sections of this Technical Report discuss the further implementation of the DSAP software infrastructure. This work includes implementing the dynamic situational awareness piece of DSAP in support of operations while also further developing the predictive analysis functionality. The remainder of this report discusses the following items:

- Section 2.0 discusses predictive capabilities for DSAP operations
- Section 3.0 discusses the design, modification, and implementation of a Multiple Replication Framework (MRF) to support dynamic situational awareness support for operations.
- Section 4.0 discusses the past efforts on DSAP development.
- Section 5.0 details work performed on this effort to transition DSAP for use on the Global Information Grid (GIG).
- Section 6.0 outlines future work.
- Section 7.0 provides the Bibliography for this Final Report.
- Section 8.0 defines the acronyms used in developing this report.

2.0 The DSAP Framework and Past Efforts

The following describes the past efforts on DSAP work that focused on areas such as implementing the predictive analysis components and implementing the dynamic situational awareness component that utilized both a real-time simulation component and a real-time calibrated simulation component. This section discusses the DSAP Concept of Operation, the DSAP MultiReplication Framework, and past updates to the framework that have been implemented as of the start of this effort.

2.1 DSAP Concept of Operations

The Conceptual Operation of the DSAP Infrastructure supports real-time data calibration, real-time state estimation through simulation, and predictive simulation through faster-than-real-time simulation. The concepts of operation for the DSAP Infrastructure are shown in Figure 2-1 with respect to the Dynamic Situation Assessment capability and Figure 2-2 with respect to the Prediction capability. In Figure 2-1, the $x_p(t)$ axis simulates the current plan in real-time. This real-time simulation is used to simulate the results and internal state of the operational picture. As this real-time simulation evolves, it is constantly being updated with Blue and Red Force information from C4I data feeds and databases. These real-time updates, denoted by the $z(t)$ axis, are provided to the real-time simulation of the current plan to calibrate the behavior of the simulated COA. This calibrated real-time simulation is used to estimate the state of the real-time operational picture, $x_e(t)$. This allows us to store the internal state, $x_e(t)$ of the mission in a manner that provides our Dynamic Situation Assessment capability.

Figure 2-2 illustrates the predictive capability of DSAP. The individual plans, $y(t)$, can be idealized and executed out in time. This basically represents the behavior of the plan when the plan executes as expected. These same plans and their alternatives are then simulated faster-than-real-time, as denoted by the $x(t)$ axis. By executing these plans faster-than-real-time, we provide a predictive look into how a plan and its execution may unfold. Multiple plans and multiple replications of each plan may be executed to provide a statistically significant outlook of a plan's anticipated outcomes based on the current operational information. This provides the Prediction capability.

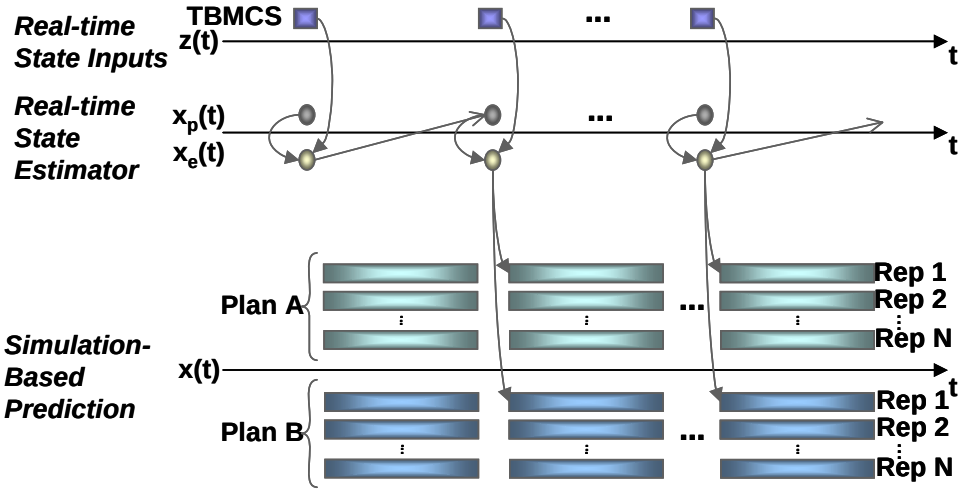

Figure 2-1: Calibrated Real-time Simulation for Estimating Operational State

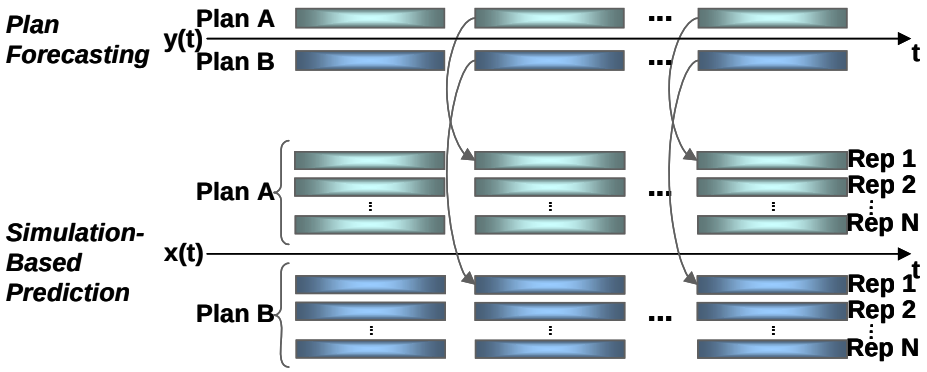

Figure 2-2: Predictive Plan Assessment through Faster-than-real-time Simulation

2.2 DSAP MultiRep Implementation at Effort Start

The basic of MRF implementation had three types of components: Taskers, Workers, and a Manager. Taskers were responsible for tasking the manager with applications for workers to run, and specify how these applications should be initialized. Workers were instructed to complete the tasks assigned by the manager, including executing the simulation replications, saving the results of the replications, and evaluating and comparing the results of the replications to the plan objectives and real-time picture. The manager's function was to divide the replications into smaller time segments, assign these tasks to the workers, handle the bookkeeping, and tackle flow control issues. Each of these components and their connectivity is shown in Figure 2-3.

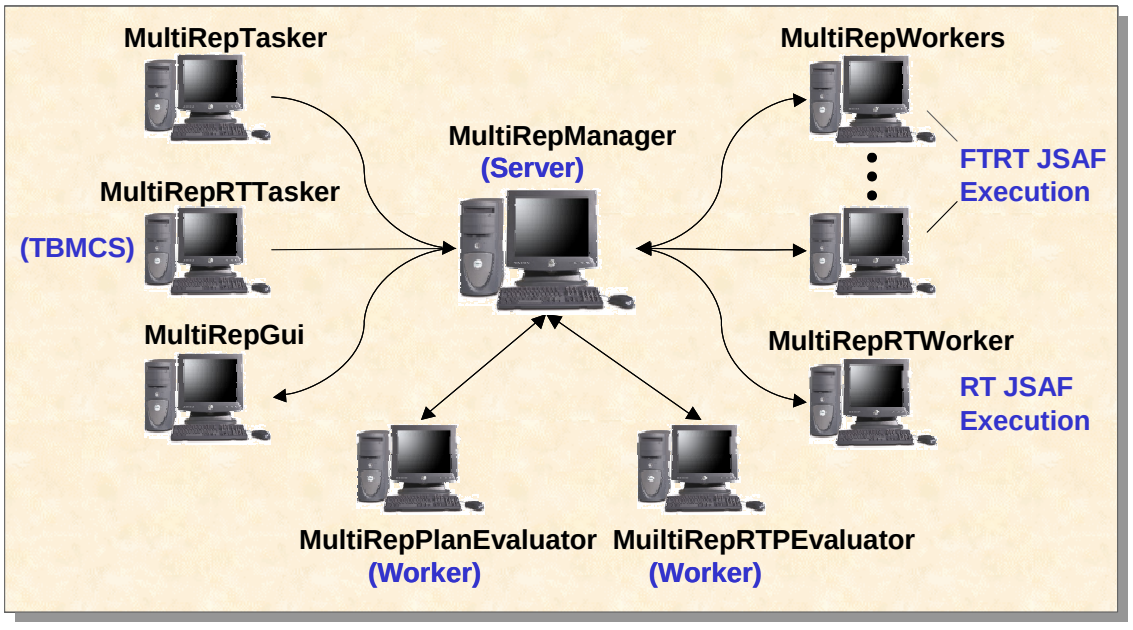

Figure 2-3: The Initial MRF

2.2.1 MultiRepTasker

The *MultiRepTasker* component was designed to interface with Command and Control to send a predictive simulation task to the server. The *MultiRepTasker* provided connectivity to the server

and allowed the user to specify initialization parameters such as the simulation execution name, initial scenario file, start and end time, simulation scaling rate, and replication number.

2.2.2 MultiRepRTTasker

The *MultiRepRTTasker* component was designed to issue a task to the server in order to initiate the real-time worker. The *MultiRepRTTasker* functions were to provide connectivity to the server and allow the user to specify the simulation execution name, initial scenario file, name of the plan the task corresponds to, and the time interval between saving the state of the simulation.

The Real-Time Tasker component, *MultiRepTBMCSTasker*, was designed to provide the capability to allow the user to retrieve the Real-Time Picture (RTP) from TBMCS or another C4I data source via command line or GUI. The sequence diagram defining the operation of the *MultiRepTBMCSTasker* is shown in Figure 2-4.

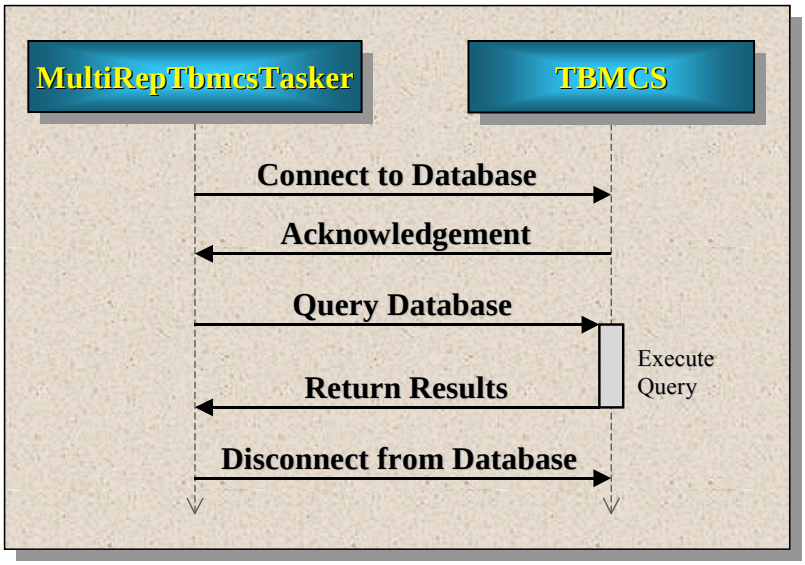

Figure 2-4: Sequence Diagram for MultiRep TBMCSTasker and TBMCS

The *MultiRepTBMCSTasker* was designed to be a Tasker component that automates the process of retrieving real-time C4I information from TBMCS. For TBMCS connectivity, both ODBC and JDBC were tried. Some problems existed with ODBC and had not been resolved. The Tasker/JDBC approach to TBMCS connectivity had been implemented and tested.

2.2.3 MultiRepManager

The server, or *MultiRepManager*, component was designed to be the core of the MRF. The *MultiRepManager* was designed to be responsible for 1) managing the execution of long replications by splicing them in time, 2) constructing the necessary parameters needed for a worker to launch and save a JSAF execution, 3) constructing the necessary parameters for launching an evaluation on an evaluator component, 4) displaying diagnostics related to the execution of multiple replications, 5) identifying when replications are completed, 6) pruning and re-tasking replications that are off course from the real-time picture, and 7) restarting unfinished replications in the event of a worker crash or disconnect.

The *MultiRepManager* design was based on the *WpServer* used to implement the Extensible Grid. The Server capability for this effort inherited from the WpNetGridServer, which was the

server for the Extensible Grid, which in turn inherited from WpServer, which was the basic server capability. The UML Class Diagram for the Server design is shown in Figure 2-5.

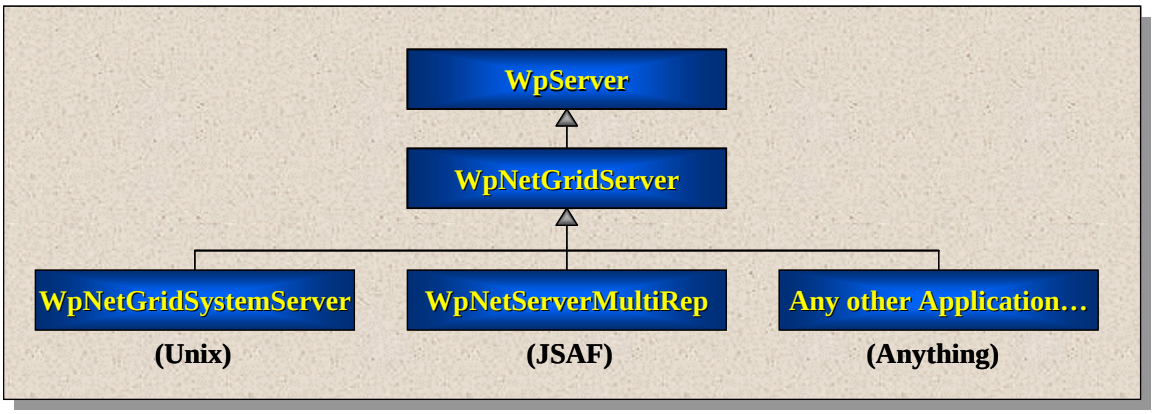

Figure 2-5: UML Diagram for Server Component Used for the MultiRepManager

The *MultiRepManager* was designed to be responsible for managing the execution and evaluation of the replications. The Activity Diagram for the *MultiRepManager* with respect to the *ProcessRtp()* function is shown in Figure 2-6. This Activity Diagram defines the process for managing the replications through their RTP evaluation.

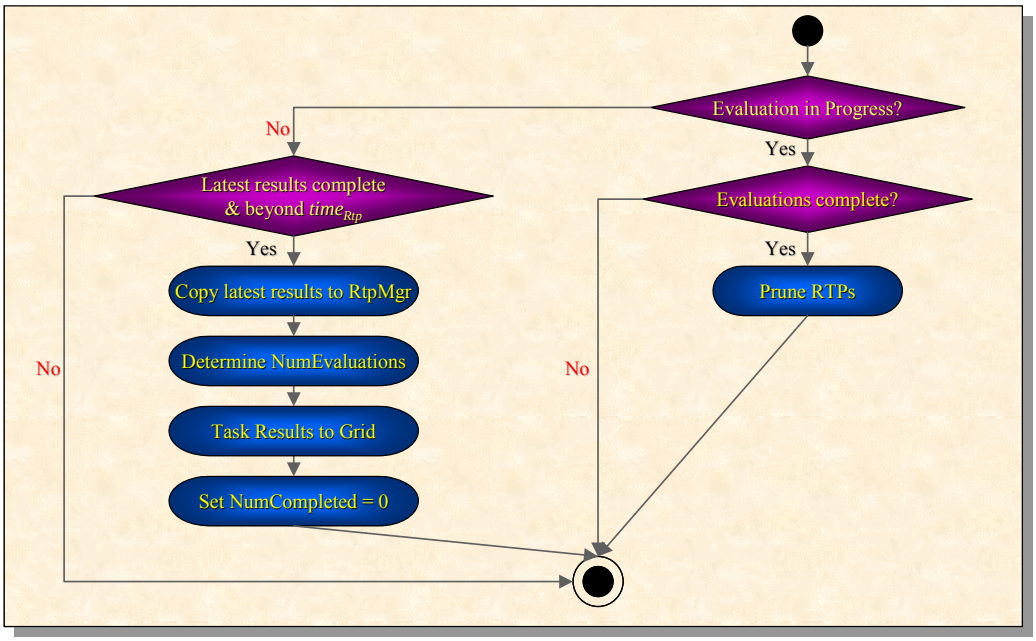

Figure 2-6: Activity Diagram for MultiRepManager's ProcessRtp()

2.2.4 MultiRepGui

The *MultiRepGui* component was designed to provide the user with diagnostics with respect to operation of the MRF. The *MultiRepGui* also allowed the user to monitor the status of the MRF. The Sequence Diagram for the *MultiRepGui* is shown in Figure 2-7. In addition to simply monitoring status and setting the time interval for faster-than-real-time simulations, the *MultiRepGui* had been modified to host our GUI. The *MultiRepGui* queried the server to return the status of replications, provided functionality to modify time intervals and end times, provided

functionality to modify pruning thresholds, and provided the capability to allow the user to prune replications or plans using the GUI.

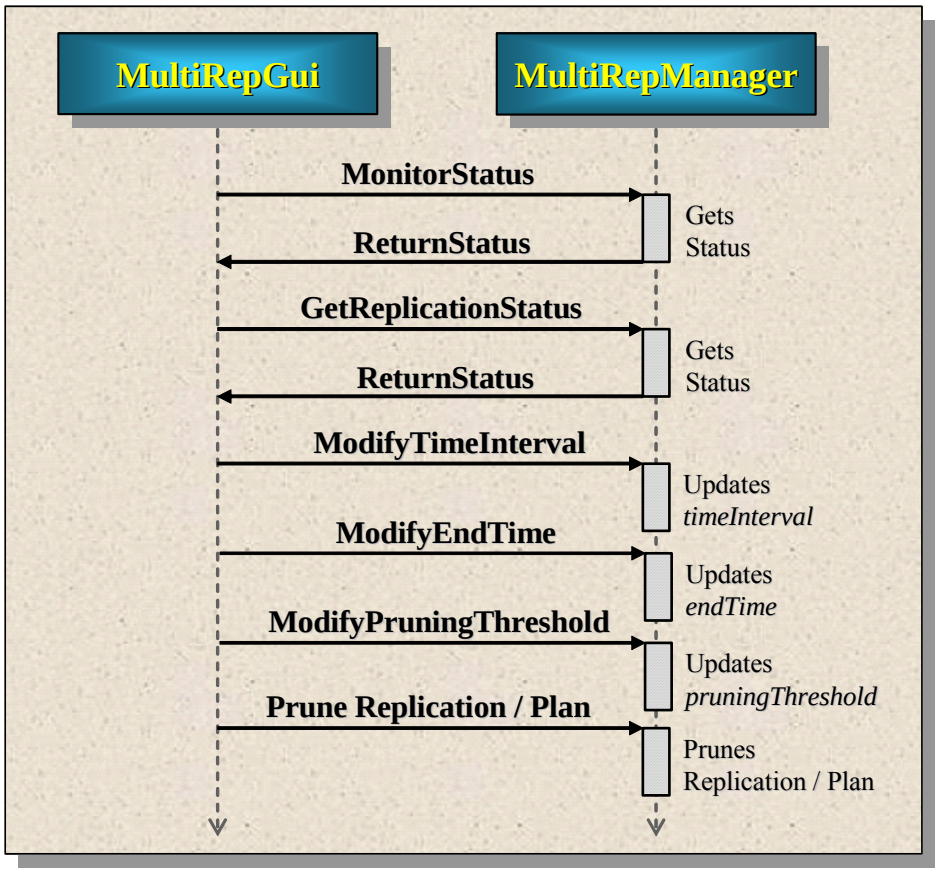

Figure 2-7: MultiRep Gui Control Flow

2.2.5 MultiRepWorker

The Worker component, *MultiRepWorker*, was designed to receive simulation tasking from the server, launch predictive JSAF executions that run faster-than-real-time, and launch JSAF replications faster-than-real-time for a predetermined length of time. The Worker received a command from the *MultiRepManager* to launch a JSAF execution which was accompanied by parameter sets (specifying the *SimRate*, start time, end time and other variables), environment variables and scenario spreadsheets. Also, the Worker was designed to pack up results from the replication execution in spreadsheet format and send the information back to the *MultiRepManager*.

Upon completion of the replication, the Worker was designed to save the state of the simulation to disk and send it to the server for later evaluation and comparison with real-time data and the plan objectives. Replications that stray from the real-time picture were automatically pruned, re-tasked by the server, and initialized to match the current state. Replications that failed to meet the plan objectives could be manually pruned by Command Staff, and if pruned, they were automatically re-tasked by the server and initialized to match the current state.

The Sequence Diagram specifying the operation of the Worker executing faster-than-real-time JSAF scenarios is shown in Figure 2-8 with respect to the rest of the MRF. This is the heart of the predictive capability for DSAP/MRF.

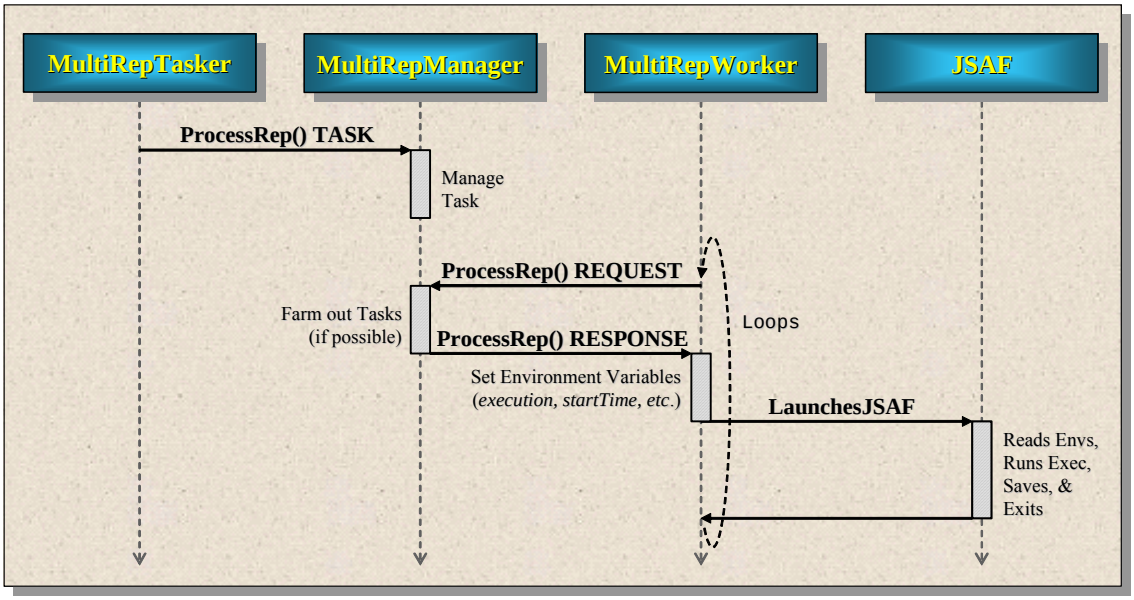

Figure 2-8: MultiRep Worker Control Flow for Executing Faster-Than-Real-Time Simulations

At update time boundaries, the real-time simulations running on *MultiRepWorker* components were synchronized with the real-time picture (in this case, live or emulated data from the Theater Battle Management Core System's (TBMCS) MIDB and AODB, which was updated every fifteen minutes via subscription). The *MultiRepRTTasker* or *MultiRepTBMCSTasker* client fed real-time information from TBMCS to the system and initiated the update process.

2.2.6 Plan Evaluator

The Plan Evaluator component, *MultiRepPlanEvaluator*, was designed to compare the state of the saved faster-than-real-time replications with the plan objectives. The *MultiRepPlanEvaluator* was designed to be a Worker that evaluated the results of the JSAF replication executions against other results. The *MultiRepPlanEvaluator* took each of the result spreadsheets and evaluated them to determine the "best" plan. The evaluation was performed by executing the function *PlanEvaulator()*. The control flow for the *MultiRepPlanEvaluator* is shown in Figure 2-9, when considering the sequence of operations between the *MultiRepPlanEvaluator*, *MultiRepManager*, and *MultiRepWorkers*. The *MultiRepPlanEvaluator* was responsible to request tasks from the server. When results spreadsheets were available at the *MultiRepManager*, those results were tasked to the *MutliRepPlanEvaluator*, which evaluated the effectiveness of each plan. The effectiveness results were then sent back to the *MultiRepManager*, and the *MultiRepRTPEvaluator* was also tasked to begin evaluating those results against the current real-time picture.

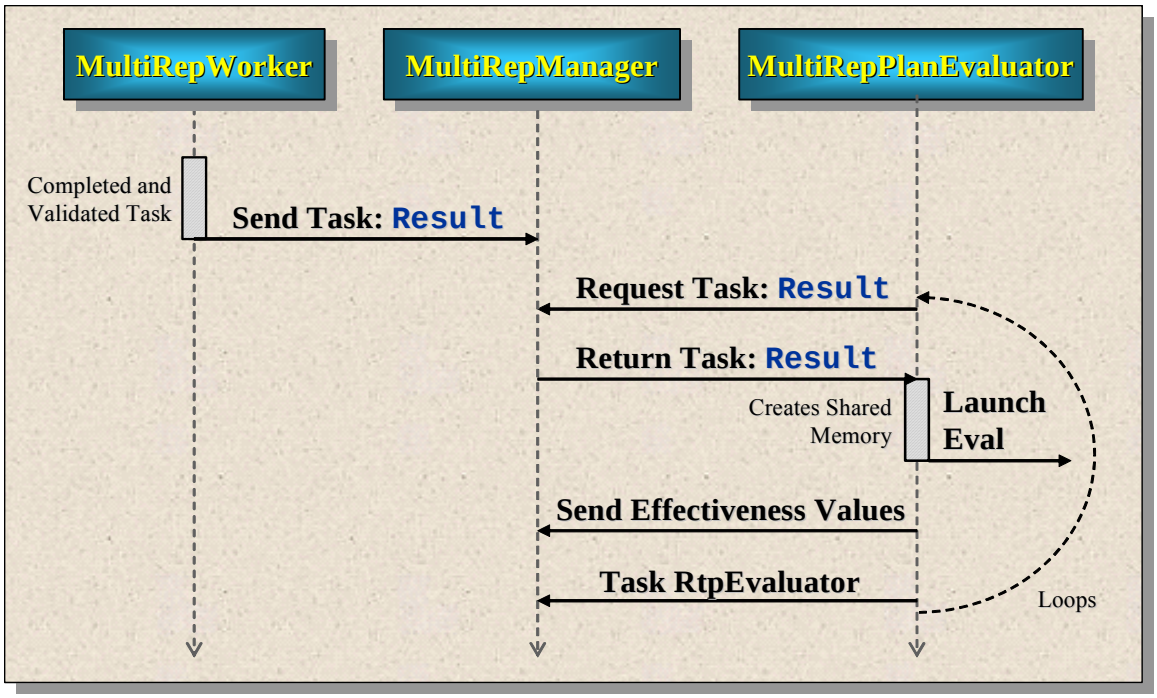

Figure 2-9: Control Flow for MultiRepPlanEvaluator

Workers (*MultiRepPlanEvaluator* and *MultiRepRTPEvaluator*) were designed to be used in evaluating the predictive simulation executions with respect to both their objectives and the real-time picture. The evaluation process was designed to compute both the *Raw Effectiveness* and the *Relative Effectiveness* of each replication. These Measures Of Effectiveness (MOEs) were used to rank each COA and determine if each plan was consistent with the real-time picture. An alternative plan might be selected or the existing COA might be maintained depending on its effectiveness. Also, if the current real-time picture showed that an alternative COA was no longer valid, that COA could be pruned or replaced by a valid alternative COA. Scenario data within the MRF's evaluation process was also updated to reflect changes in assets and resource status based on real-time picture updates. At that point, the COAs and alternates were once again farmed out and executed on the available processors.

The initial MRF GUI is shown in Figure 2-10. The GUI was designed to provide a graphical interactive interface to the MRF that allowed Commanders to task the execution of simulation plans and replications, view the progress and performance of the plans, and prune ineffective plans. The graph at the bottom of the GUI was designed to plot the raw and relative effectiveness of each plan over time. These metrics were used to gauge the effectiveness and performance of the Commander's plan.

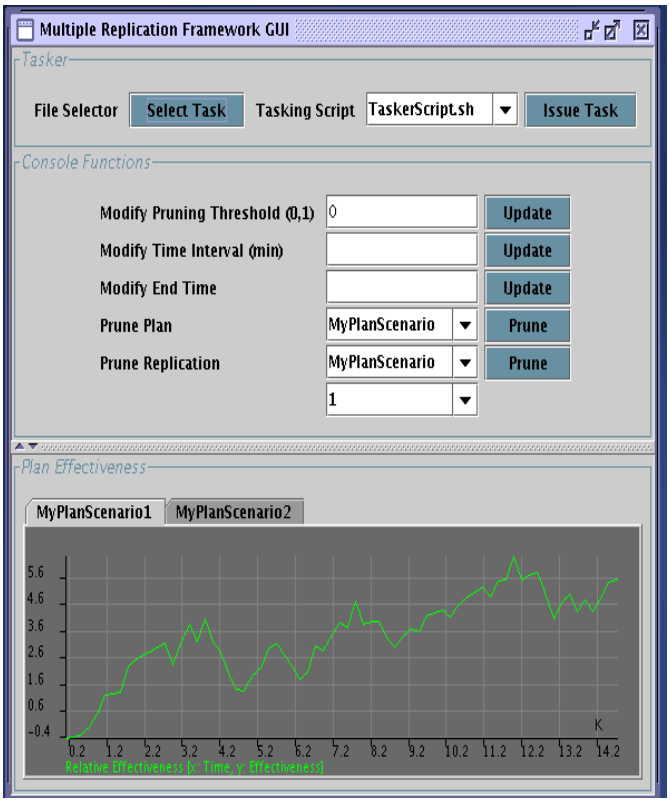

Figure 2-10: The MRF GUI

Plans and replications were initiated in the MRF by issuing a tasking script from the GUI. As shown in Figure 2-11, a file selector tool was designed to allow the Commander to select a tasking script to kick off the process. The GUI would be expanded to allow the Commander to start plans and replications via menus instead of scripts.

After the script had been selected, the MRF took control by sending the task to available workers, as shown in Figure 2-12.

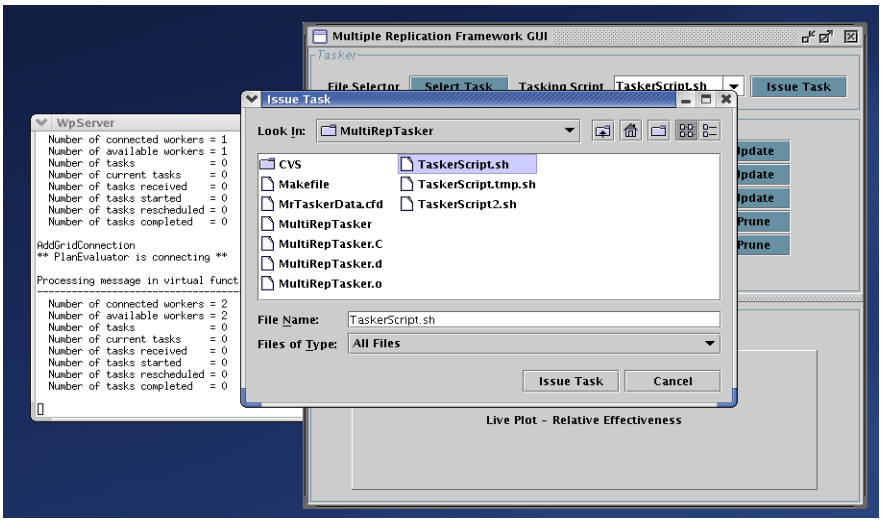

Figure 2-11: Selecting and Issuing a Tasking Script

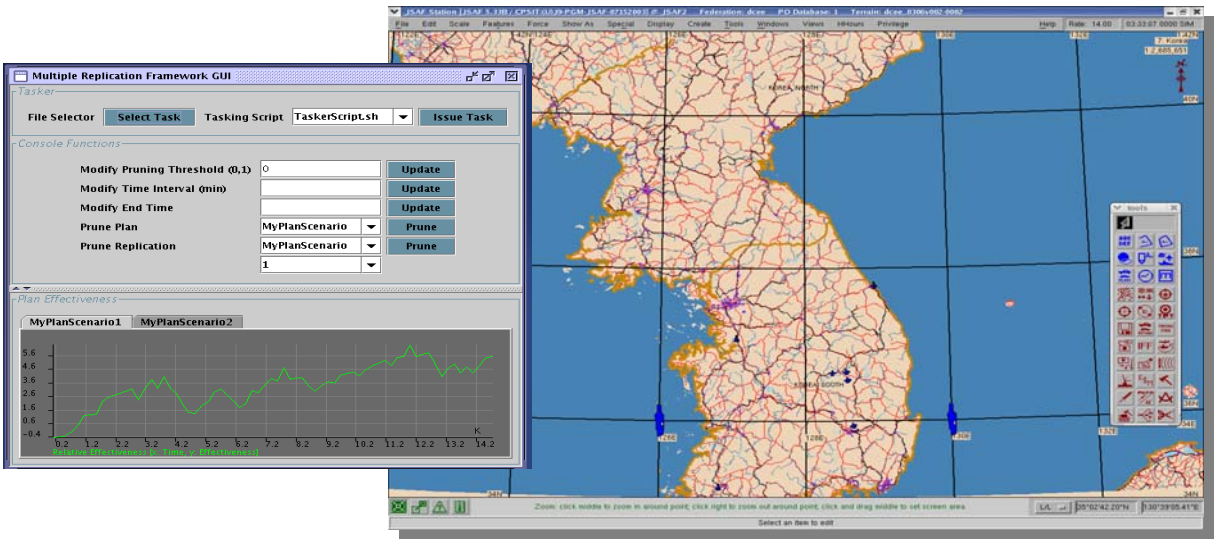

Figure 2-12: GUI kicking off a JSAF Replication

After the simulation time segment was completed, the MRF was automatically responsible to perform the plan evaluation and real-time picture evaluation, if the real-time data update was received. Figure 2-13 shows the plan evaluation results of a replication.

Because of the uncertain nature of predicting the outcome of plans, it was important to execute multiple replications of a plan and statistically analyze the results. The MRF was responsible to calculate the mean and standard deviation of the effectiveness values for each time step in the simulation.

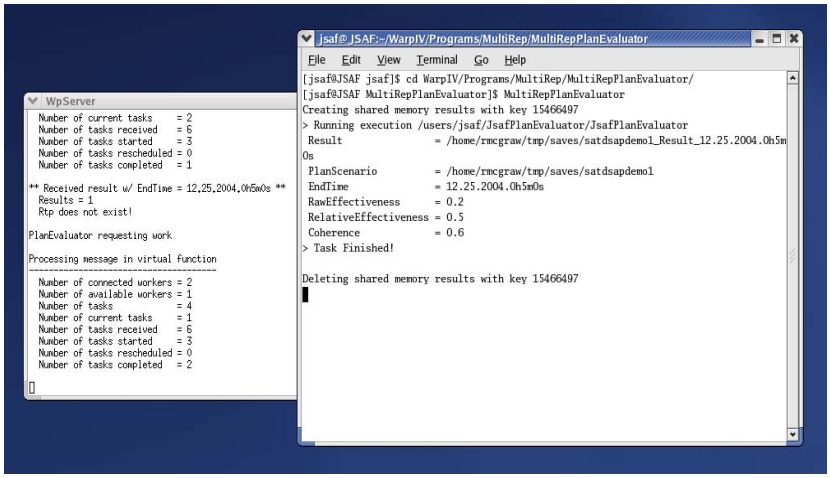

Figure 2-13: Plan Evaluation Results

2.3 Updates to the Initial MRF

The MRF discussed in Section 2.2 provides a predictive analysis capability for C2 through the use of embedded simulations. Further development was made to the DSAP MRF to provide a real-time dynamic situational awareness capability on a subsequent effort. These enhancements entailed using the MRF to simulate the current plan in real-time using two real-time simulations:

a real-time simulation of the idealized plan, and a real-time simulation that is constantly calibrated with TBMCS inputs. These modifications to the MRF architecture, topology, data flow and sequence of operations are discussed in the following sections.

The effort updated the MRF as shown in Figure 2-14. Here, a Real-time Worker (*MultiRepRTWorker*) and a Real-Time Calibrated Worker (*MultiRepRTCWorker*) are added to the framework. The *MultiRepRTWorker* component is responsible for executing the idealized simulation of the plan using the simulation environment of choice (in our case JSAF). This simulation is not calibrated with real-time inputs. The intent is for this simulation to play out the idealized plan based on the state of operations at the plan initiation. The simulated state of operations for the idealized plan is saved and sent to the *MultiRepManager* every user-defined time segment. The implementation is designed to use JSAF's native damage reporting capability to store the plan state in comma delimited format. This idealized plan information is used to compute Real Effectiveness by the *MultiRepRTPEvaluator*.

The second component implemented is the real-time calibrated worker (*MultiRepRTCWorker*). This component is designed to receive real-time updates, and must shut down and restart to receive and reflect the updates. This process was implemented using JSAF's checkpoint and restarted capabilities. Additional work calibrates the real-time worker (and thus the JSAF simulation) without utilizing a checkpoint restarted process. This approach allows the worker to receive and reflect the updates while the simulation.

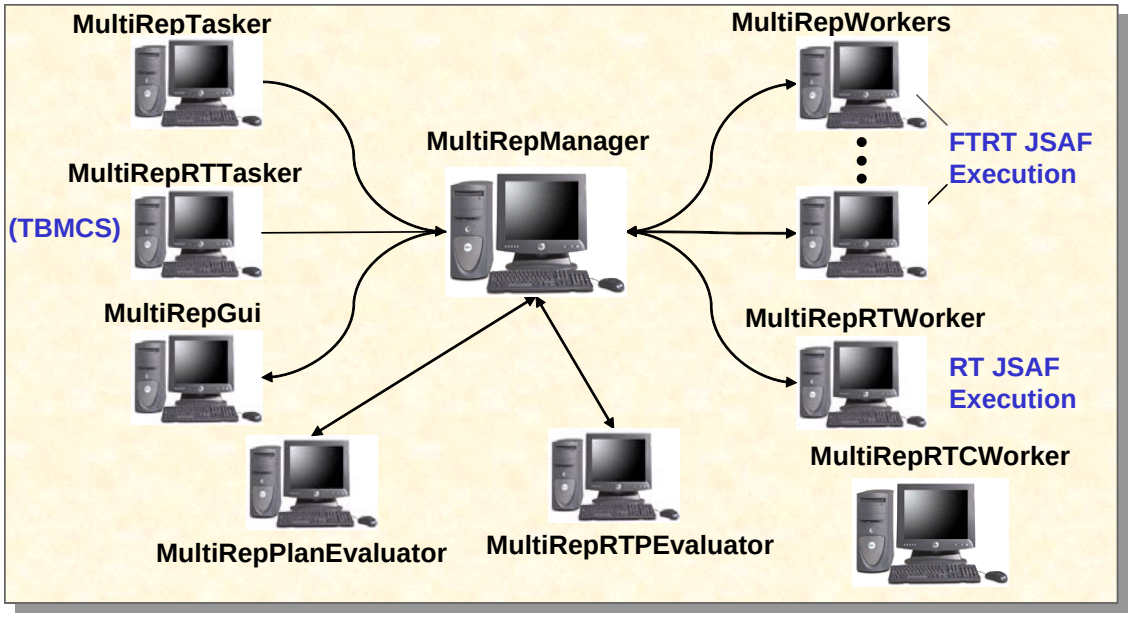

Figure 2-14: MRF Updates for Dynamic Situational Awareness Capability

2.3.1 Updates to the MultiRepRTPEvaluator

To implement the real-time capability in support of state estimation, a *MultiRepRTPEvaluator* component has been implemented that ensured the RTP Evaluator compared the idealized plan (simulated on *MultiRepRTWorkers*) with a calibrated real-time plan (simulated on *MultiRepRTCWorkers*). This approach results in several updates to the control flow associated with the *MultiRepRTPEvaluator*. The first update, shown in Figure 2-15, adds functionality for looping through RTP evaluation results, and pruning replications that exceed some (user-

specified) threshold. This modified control flow also assumes that the *MultiRepRTPEvaluator* has connected to the server. The second modification to this control flow, shown in Figure 2-16, shows the modified control flow based on changes made to the *MultiRepTasker* components which combine the functionality of the emulated TBMCS and TBMCS real-time calibration process, which is emulated using a flat-file or spreadsheet.

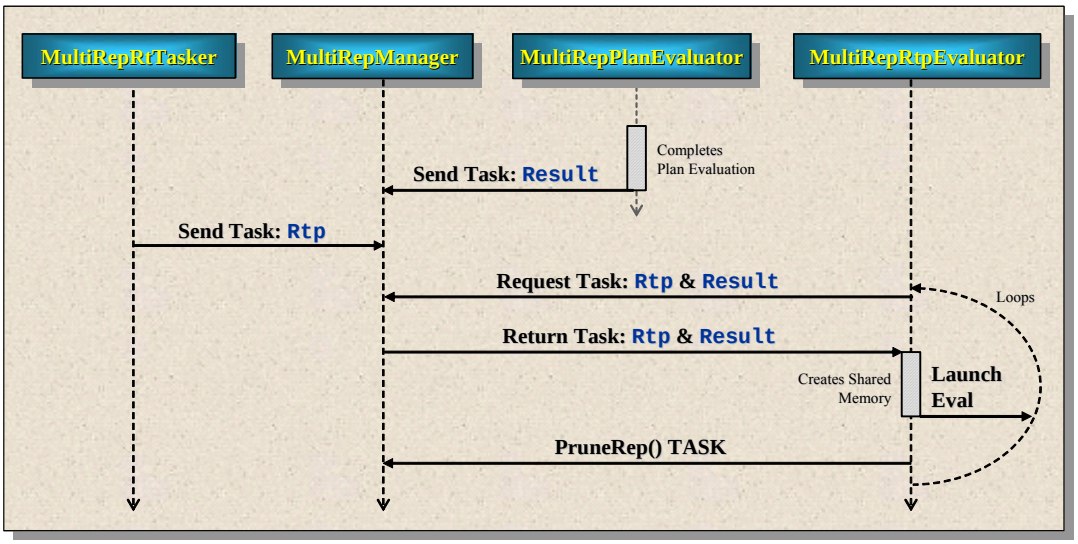

Figure 2-15: RTP Evaluator Modification To Prune Replications

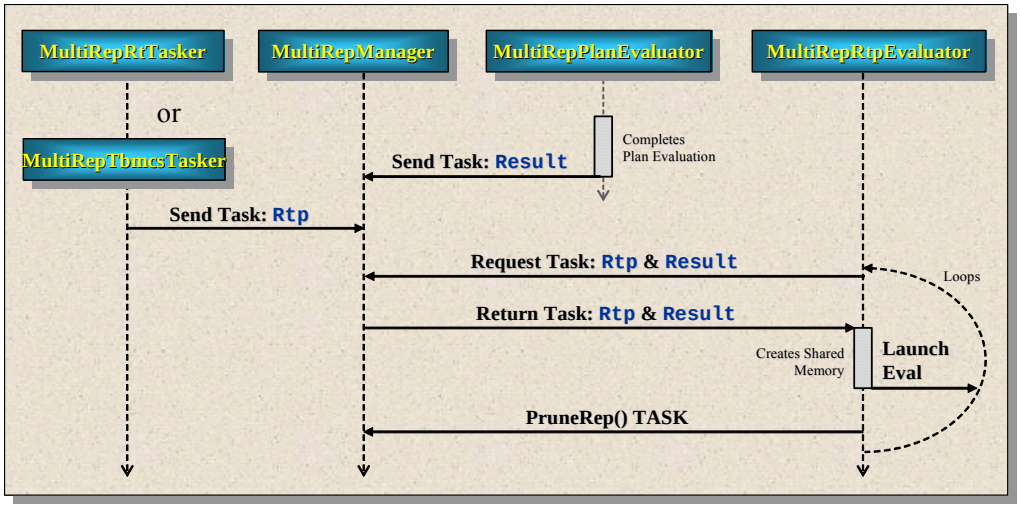

Figure 2-16: RTP Modification to Reflect Tasker Enhancements

2.3.2 MultiRepRTCWorker

A *MultiRepRTCWorker* is used to run real-time JSAF executions (or executions of other simulations) in order to support the state estimation capability in the MRF. The *MultiRepRTCWorker* is designed to be responsible for running the real-time simulation, updating the simulation with TBMCS information (in the case of the calibrated real-time simulation), and saving checkpoints of the simulation to intermediate results files for effectiveness evaluations. The sequence diagram of the *MultiRepRTCWorker* is shown in Figure 2-17. For the

MultiRepRTCWorker, the worker is designed to connect to the server and request tasking. The Tasking, when provided to the server is designed to assign tasks to the server that are passed to the *MultiRepRTCWorker*. The *MultiRepRTCWorker* is responsible to execute JSAF tasks and send results back to the Server every 15 minutes. In addition, the *MultiRepRTTasker* is continually responsible to update the real-time simulation every 15 minutes.

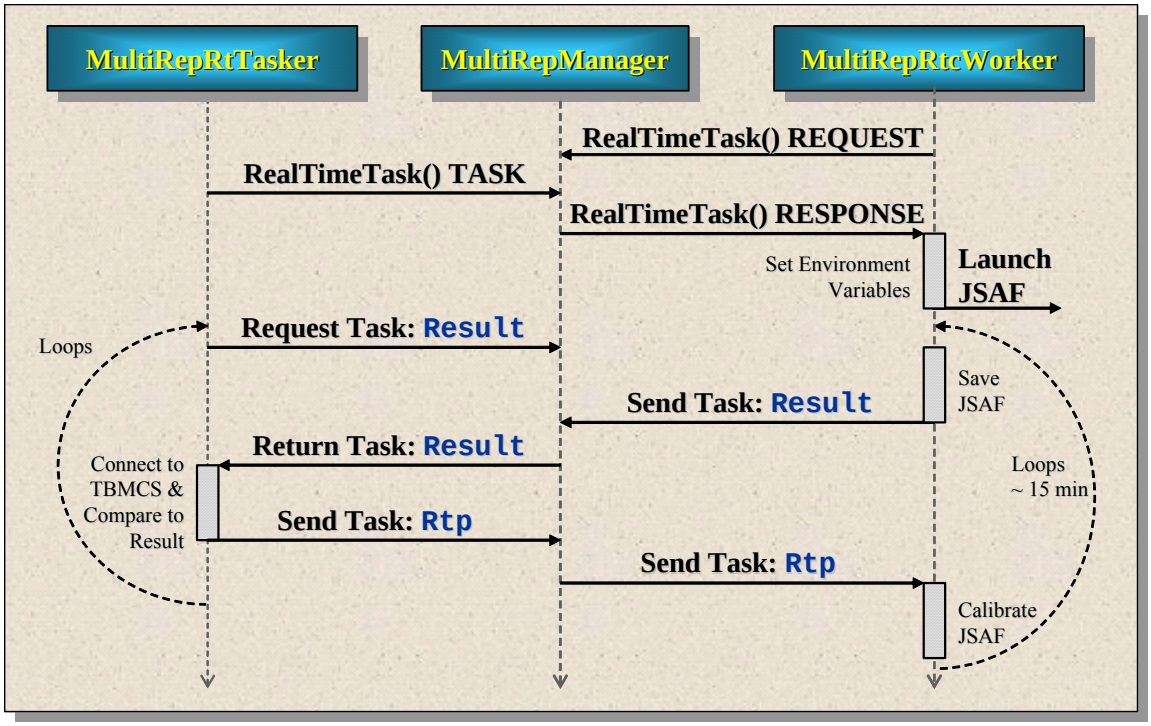

Figure 2-17: Sequence Diagram for MultiRepRTCWorker

2.3.3 MultiRepRTWorker

The *MultiRepRTWorker* component is designed to execute the idealized plan in real-time. The *MultiRepRTWorker* is responsible to behave like the *MultiRepRTCWorker* and *MultiRepWorker* with the exception that the *MultiRepRTWorker* is not responsible to calibrate with real-time data. The *MultiRepRTWorker* saves its state every 15 minutes and sends this information to the server. The *MultiRepRTWorker* remains up (does not shut down or terminate) during the execution of the MRF. The sequence diagram of the *MultiRepRTWorker* is shown in Figure 2-18.

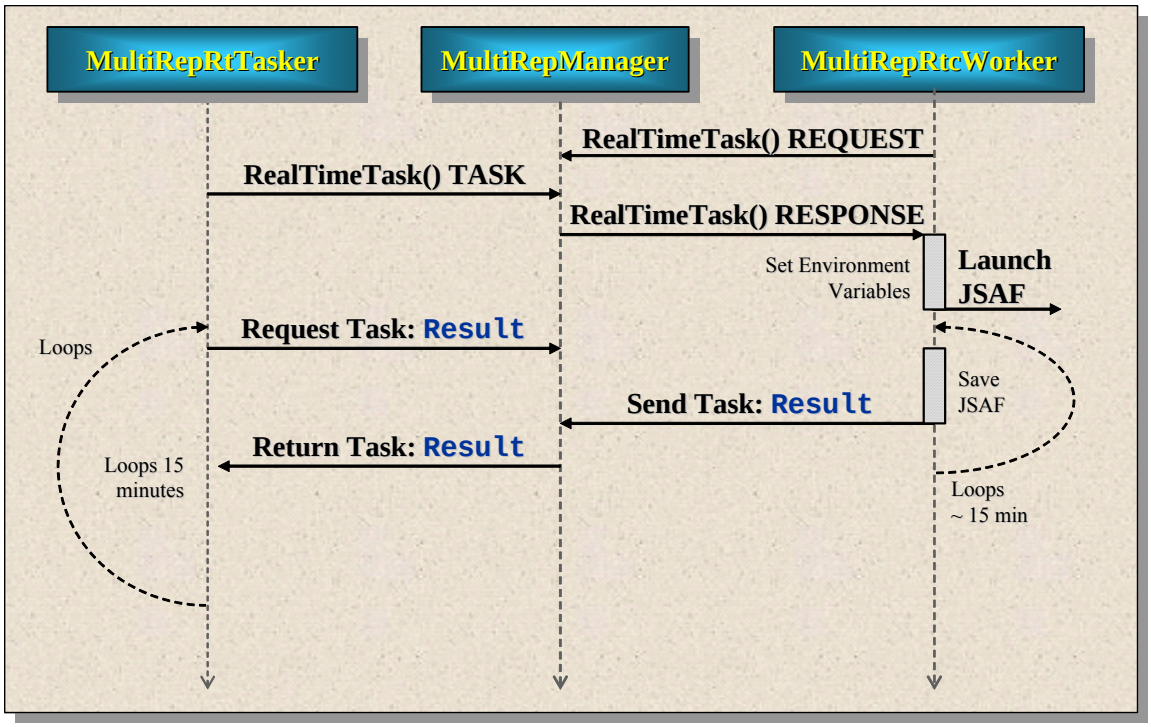

Figure 2-18: Sequence Diagram for MultiRepRtWorker

Once these components were implemented, the control for this dynamic situation assessment capability was defined and built within the MRF to handle the Real-Time Worker and Real-Time Calibrated Worker components. The sequence diagram for these capabilities is shown in Figure 2-19.

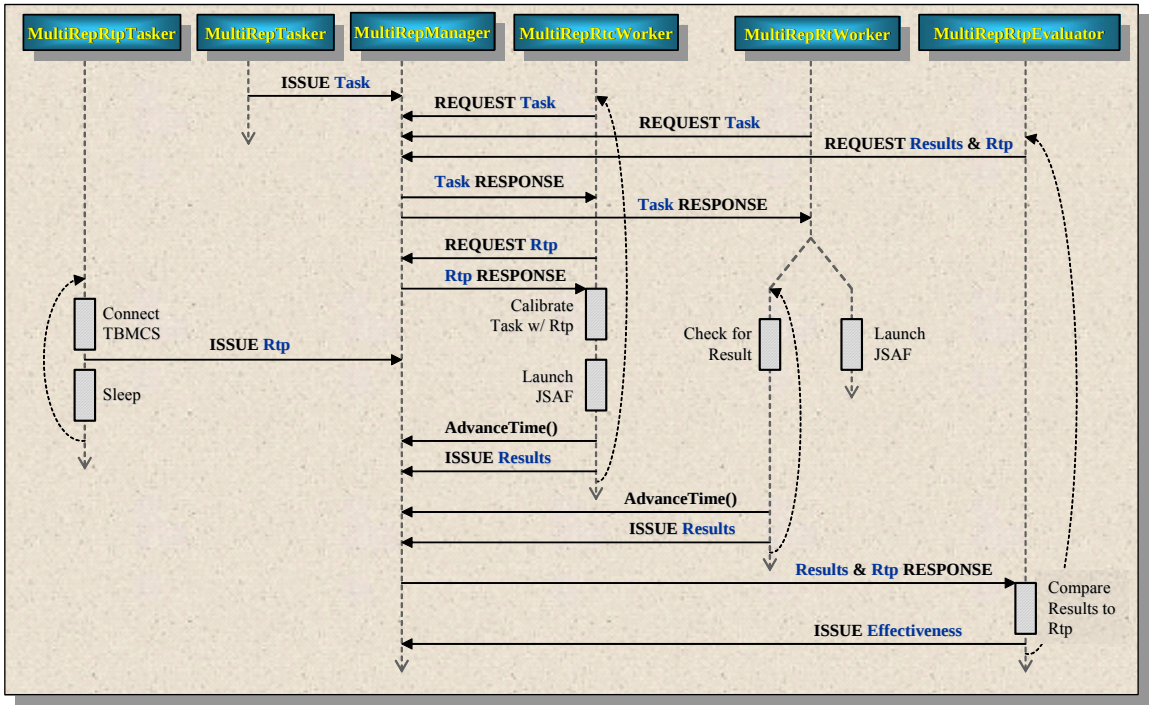

Figure 2-19: Dynamic Situation Assessment Sequence Diagram

2.4 Extracting Damage Results

In addition to developing processes for calibrating real-time simulations using TBMCS, our past efforts also took steps to store damage and operational state information for the running JSAF simulations to use in the Raw and Real Effectiveness calculations determined by the *MultiRepPlanEvaluator* and *MultiRepRTPEvaluator* components. This can be accomplished by adding to the JSAF source code as highlighted in the snippet box below.

```
void SaveUnitStatus(char *fileName, PO_DATABASE *po_db) {

  if (!po_db) {
    cout << "ERROR: po_db does not exist!" << endl;
    return;
  }

  ofstream outFile(fileName);
  if (!outFile) {
    cout << "ERROR: file " << fileName << " could not be opened!" << endl;
    return;
  }

  cout << "Saving Unit Status..." << endl;
  int i;
  PO_DB_ENTRY *entry;
  time_pause(po_db->save_load_pause_handle);
  outFile << "CALLSIGN,UNIT TYPE,POSITION,LAT,LON,Z,TOTAL,CAPABLE,DAMAGED,DESTROYED"
          << endl << endl;

  for (i=0; i<PO_MAX_OBJECT_CLASS; i++) {
    for (entry = po_db->object_classes[i]; entry; entry = entry->next_class) {
      if (!entry->implied && !entry->deleted) {
        string callsign = (char *)PO_UNIT_DATA(entry).marking.text;
        string type = stdname_get_standard(
                protocol_find_name(PO_UNIT_DATA(entry).objectType));
    if (!callsign.empty() && type != "Constant Unknown") {

      struct usg_sub_counts usc;
      usc.total = 0;
      usc.capable = 0;
      usc.damaged = 0;
      usc.destroyed = 0;

      unitorg_traverse_tree(entry,
                    TRUE,
                    UsgAddVehicle,
                    (UNITORG_FUNCTION_ARG)&usc);

      LatLonCoordinates ll = GcsToLatLon(PO_UNIT_DATA(entry).location);

      outFile << callsign << ","
            << type << ","
            << UsgGetLocString(PO_UNIT_DATA(entry).location) << ","
            << ll.Lat << ","
            << ll.Lon << ","
            << ll.Z << ","
            << usc.total << ","
            << usc.capable << ","
            << usc.damaged << ","
            << usc.destroyed << endl;
    }
    }
  }
}
```

Code Segment 2-1: Code for Extracting Damage and State Results from Running JSAF Simulations

3.0 Updates to Predictive Operations for Exercise and Experimentation Support

RAM Laboratories has developed the DSAP Framework to address predictive analysis of plans and Courses of Action in order to provide inputs to the operational C2 environment at Air Operations Centers. This final report reflects on efforts to build on the DSAP Framework and its Multiple Replication Framework (MRF), while enhancing its underlying infrastructure. These enhancements seek to provide a dynamic situational awareness capability that can be eventually used to support exercises and experiments, in addition to working with 3rd party planning and analysis tools. This section describes these updates.

3.1 Update of DSAP and MRF Implementation for Predictive Operations

The overall DSAP Framework is built on RAM Laboratories' underlying Extensible Grid technology and utilizes Object Request Broker (ORB) concepts. This technology is open source and has been developed to support a number of defense programs involving the Air Force, Navy, and Missile Defense agency. An Operational Vies (OV) of DSAP is shown in Figure 3-1.

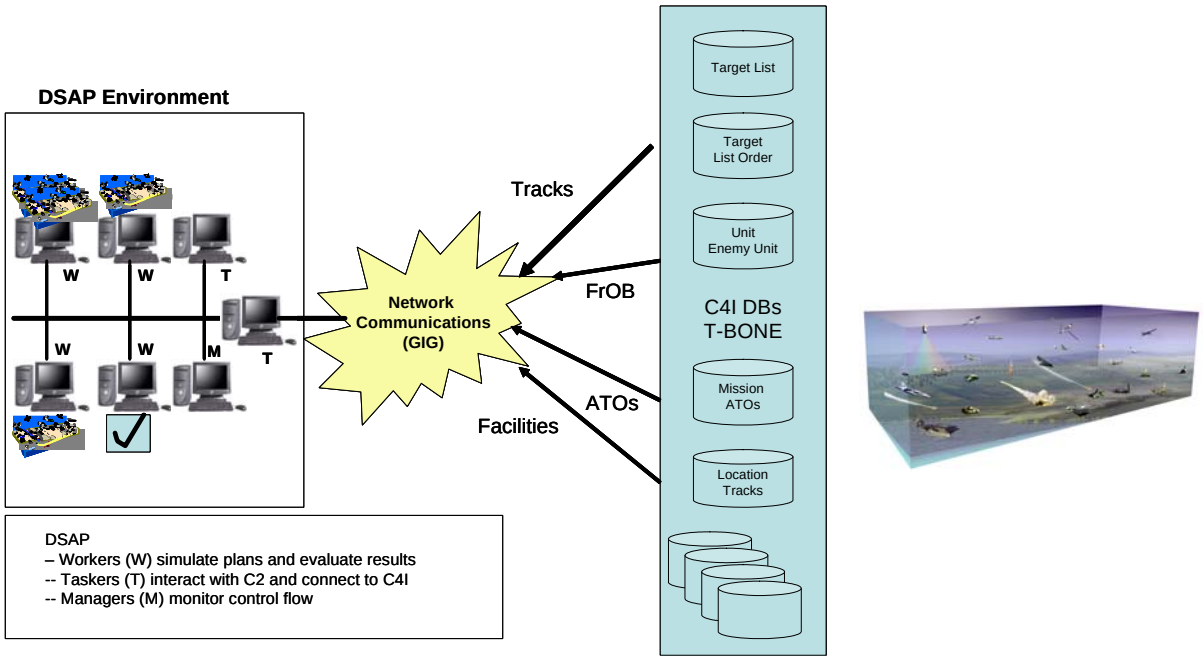

Figure 3-1: DSAP Operational View

3.1.1 Updated DSAP Framework

The DSAP Framework builds on advanced information technologies, the RAM Laboratories' Extensible Grid and CORBA technologies. The DSAP Framework and/or underlying MRF objectives are (1) building high-performance distributed computing applications, (2) integrating easily into any application domains such as Service Oriented Architecture (SOA), (3) enabling simple and effective application-level integration with a wide range of interchangeable software components, and (4) solving problem areas where multiple alternative solutions may need to be

examined in a timely manner such as (i) planning effectiveness analysis and prediction, (ii) real-time simulation for enhanced situational awareness, and (iii) plan optimization problems.

With the fundamental design intact, the MRF went through a major re-architecture in order to meet the important design goals for supporting the ever-changing, evolving software technology and promoting plug-and-play of DSAP components

These important design goals for this effort required the framework to (1) be easily adaptable to many different application requirements, (2) allow for natural decomposition of complex applications into functional components, (3) migrate responsibility for all application functionality out of server process and into client processes, and (4) enable development of applications with a rich assortment of different client agents and outside interfaces such as Web services, advanced presentation modules, and third-party software component interfaces (building System-of-Systems).

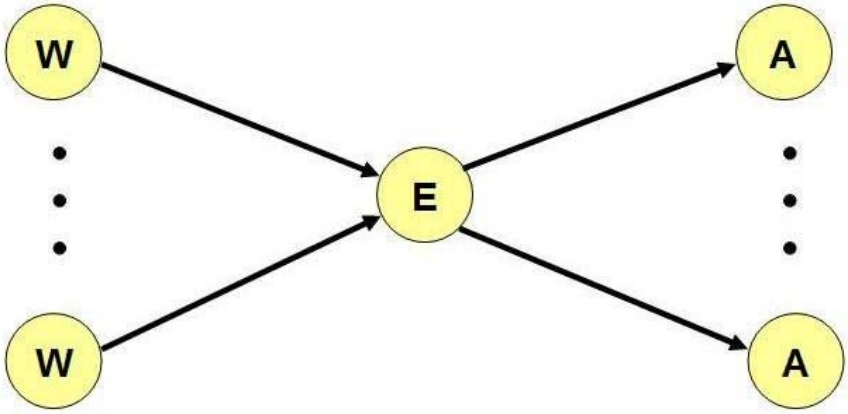

Figure 3-2: The MultiRep application pattern

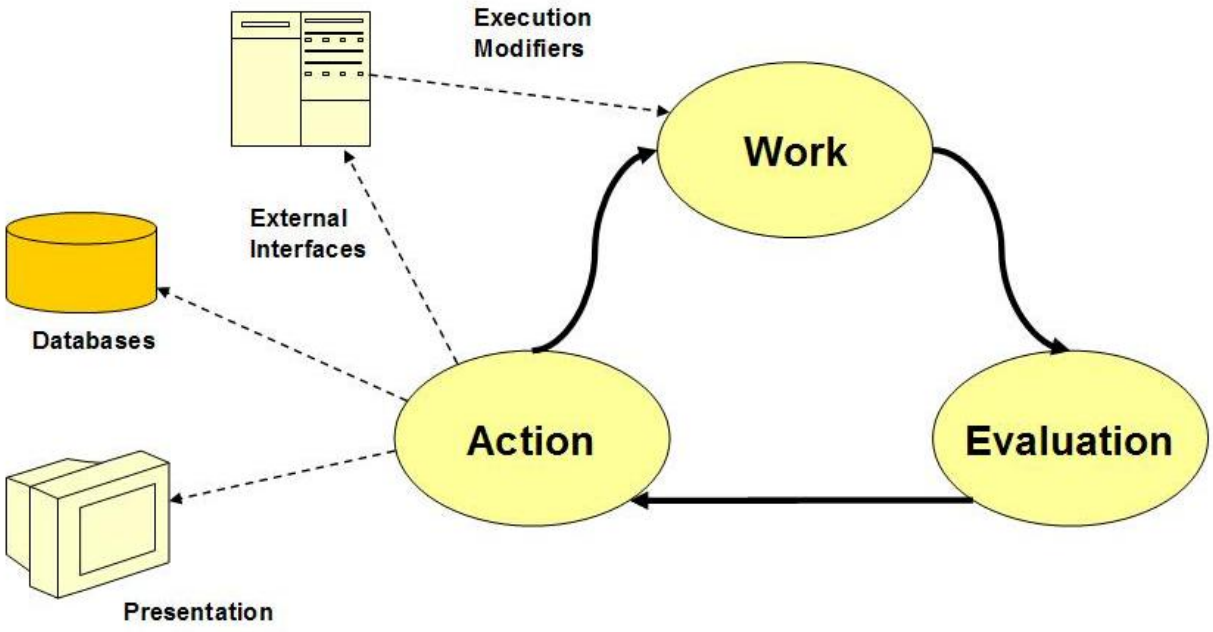

Figure 3-3: The MultiRep execution flow

Figure 3-2 illustrates the MultiRep application pattern that can be applied using the new architecture of MRF. There are three agents in the pattern of MRF: Worker, Evaluator, and Action. The Worker agents are designed to run multiple tasks and replications of tasks concurrently. The Evaluator agents (formerly a Worker component) are designed to evaluate or consolidate results and distribute reports. The Action agents (formerly Tasker components) are designed to perform periodic functions in response to reports. Also, Figure 3-3 illustrates the general MultiRep execution flow in a repeated cycle for a predictive application.

3.1.2 Updated DSAP Implementation

This DSAP prototype allows multiple replications of a simulation to run concurrently across parallel and distributed platforms, gather global statistics on the simulations upon completion, splice simulation executions into shorter runs, and compare simulation results with real-time data enabling the Commander to make decision on pruning simulations that diverge from the plan objectives.

3.1.2.1 Predictive Analysis Enhancements

In the MRF, the *MRFTasker* client functions to kick off the plan evaluation process by tasking the simulation of replications of COAs and alternate COAs to available *MRFWorkers* running JSAF faster-than-real-time. The *MRFTasker* allows the user to determine which simulation scenario to run, the scenario execution name, and the start and end time of the simulation. Objectives for the COA are also specified through the use of the console or the *DSAP GUI*. The *MRF_Manager* functions as the server in that it handles receiving, queuing, and intelligent task distribution. The *MRF_Manager* farms out replications for each plan that are run faster-than-real-time on available processors. When completed, the results of those replications are written to memory and file system and new replications can begin in that same time slot.

An overview of the current capabilities of the DSAP prototype is discussed as follows, with reference to Figure 3-4. The MRF serves to farm-out and run multiple replications of plans and alternative COAs via faster-than-real-time simulation. When real-time updates are available, state information can be calibrated with real-time C4I inputs, saved and compared with the state information from the predictive plans. Plan replications that do not meet the plan objectives they can be pruned and replaced by the Commander through a centralized controller called *Evaluator GUI*. Our MRF prototype manages this entire process by utilizing a data source for our real-time C4I inputs and Joint SemiAutomated Forces (JSAF) as both the real-time and faster-than-real-time simulation components. JSAF was selected as our simulation component because of its ability to simulate a Joint Urban Operations (JUO) environment (as well as theater operations) as well as its enhanced support for intelligent ground clutter models, which is the current focus for the Air Force sponsor. It should also be noted that other simulations can be used as the simulation component depending on the desired application.

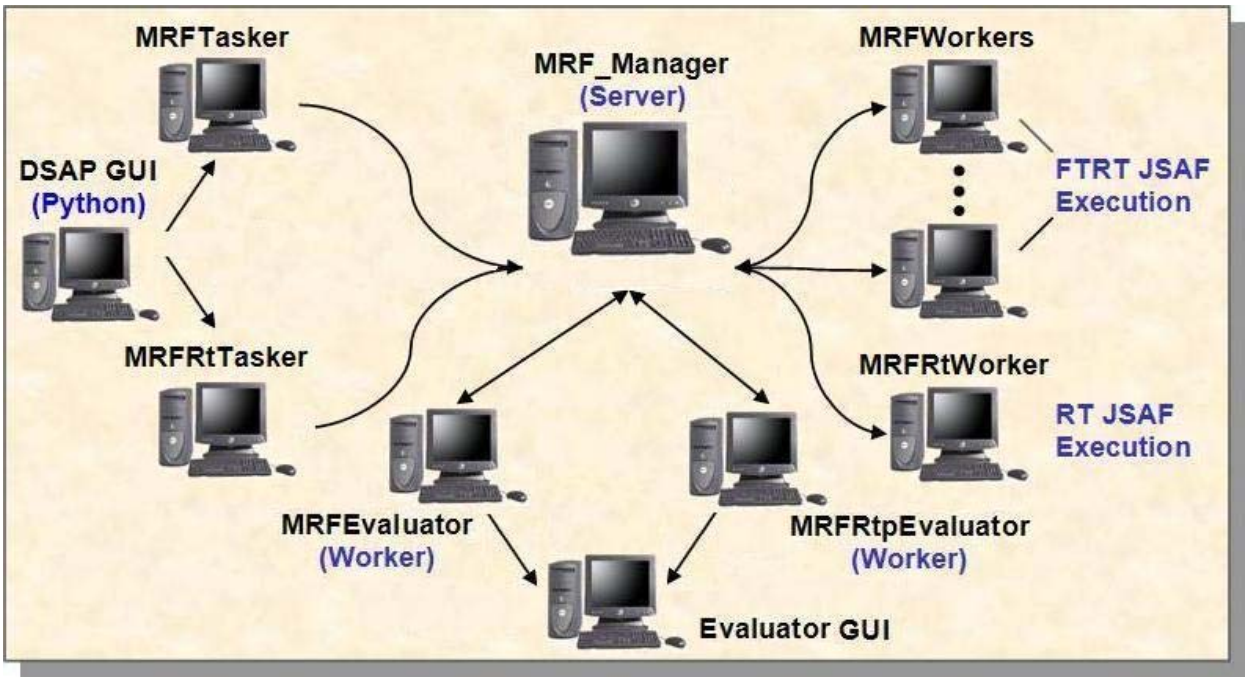

Figure 3-4: The Updated MRF for this Effort

3.2 Updated MRF Components

As before, the MRF has three basic types of components: Taskers, Workers, and a Manager. Taskers function to task the manager with applications for workers to run, and specify how these applications should be initialized, and prune unsuccessful replications or plans. These are how hosted on "Action" agents (see Figure 3-3). Two types of Workers are used. "Work" agents are Workers that complete the tasks assigned by the manager, including executing the simulation replications, and saving the results of the replications. "Evaluation" agents are Workers that are responsible for evaluating and comparing the results of the replications to the plan objectives. The manager tracks time segments of replications, assigns tasks to the workers, and tackles flow control issues. Figure 3-4 illustrates specific instances of each of these components and their connectivity with the manager. The role of each of the specific MRF components for this effort is discussed in the following subsections.

3.2.1 MRFTasker

The *MRFTasker* component interfaces with Command and Control to send a predictive simulation task to the server. The *MRFTasker* provides connectivity to the server and allows the user to specify initialization parameters such as the simulation execution name, initial scenario file, start and end time, simulation rate, and replication number.

3.2.2 MRF_Manager

The server, *MRF_Manager* or *WpMRFServer*, component is the core of the MRF. The *MRF_Manager* is responsible for (1) tracking time segments of replications, (2) constructing the necessary parameters needed for a worker to launch and save a JSAF execution, (3) constructing the necessary parameters for launching an evaluation on an evaluator component, (4) identifying

when replications are completed, (5) managing the tasking and re-tasking flow control issues, and (6) restarting unfinished replications in the event of a worker crash or disconnect.

The *MRF_Manager* design builds off of the *WpMultiRepServer* used to implement the Extensible Grid. The Server capability for this effort inherits from the WpNetGridMRFServer, which is the server for the Extensible Grid, which in turn inherits from WpNetServer, which is the basic server capability. The UML Class Diagram for the Server design is shown in Figure 3-5.

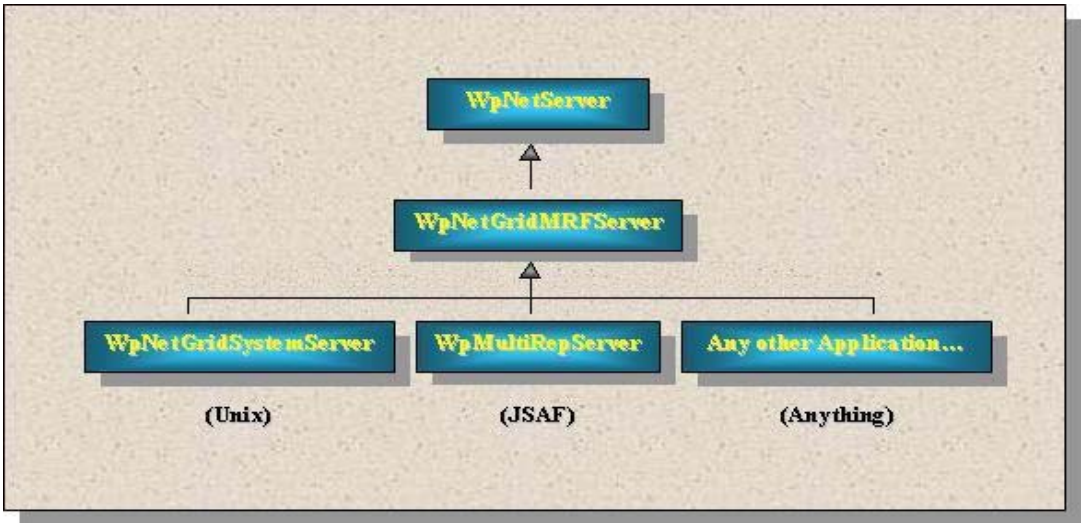

Figure 3-5: UML Diagram for Server Component Used for the MRF_Manager

Figure 3-6: Class Diagram for WpMultiRepServer

The *MRF_Manager* is responsible for tracking time segments of replications and managing the tasking and re-tasking flow control issues. The operations and attributes of the *WpMultiRepServer* class that the *MRF_Manager* inherits can be shown in Figure 3-6. All Worker application codes must be derived from *WpMultiRepClient* as shown in Figure 3-7.

Figure 3-7: Class Diagram for WpMultiRepClient

3.2.3 DSAP GUI

The *DSAP GUI* component is the front-end component in DSAP. It provides a user-friendly interface that allows the user to manage and organize the operation of the MRF. The *DSAP GUI* consists of two graphical interfaces, *MR_Startup* and *TaskApp*. These have been written in Python and are linked to DSAP libraries in order to interface with other DSAP components. As shown in Figure 3-8, the *MR_Startup* is responsible for (1) managing the categorization of servers and clients by allowing the user to define their execution properties, and (2) launching servers and clients and displaying their status and results of each execution.

As shown in Figure 3-9, the *TaskApp* complements the *MR_Startup* by giving the user the following capabilities (1) managing the categorization of tasks in the same manner as *MR_Startup* does for clients, (2) splicing simulation executions into shorter runs or smaller time segments, and (3) launching tasks individually or as a whole group and displaying their status and results.

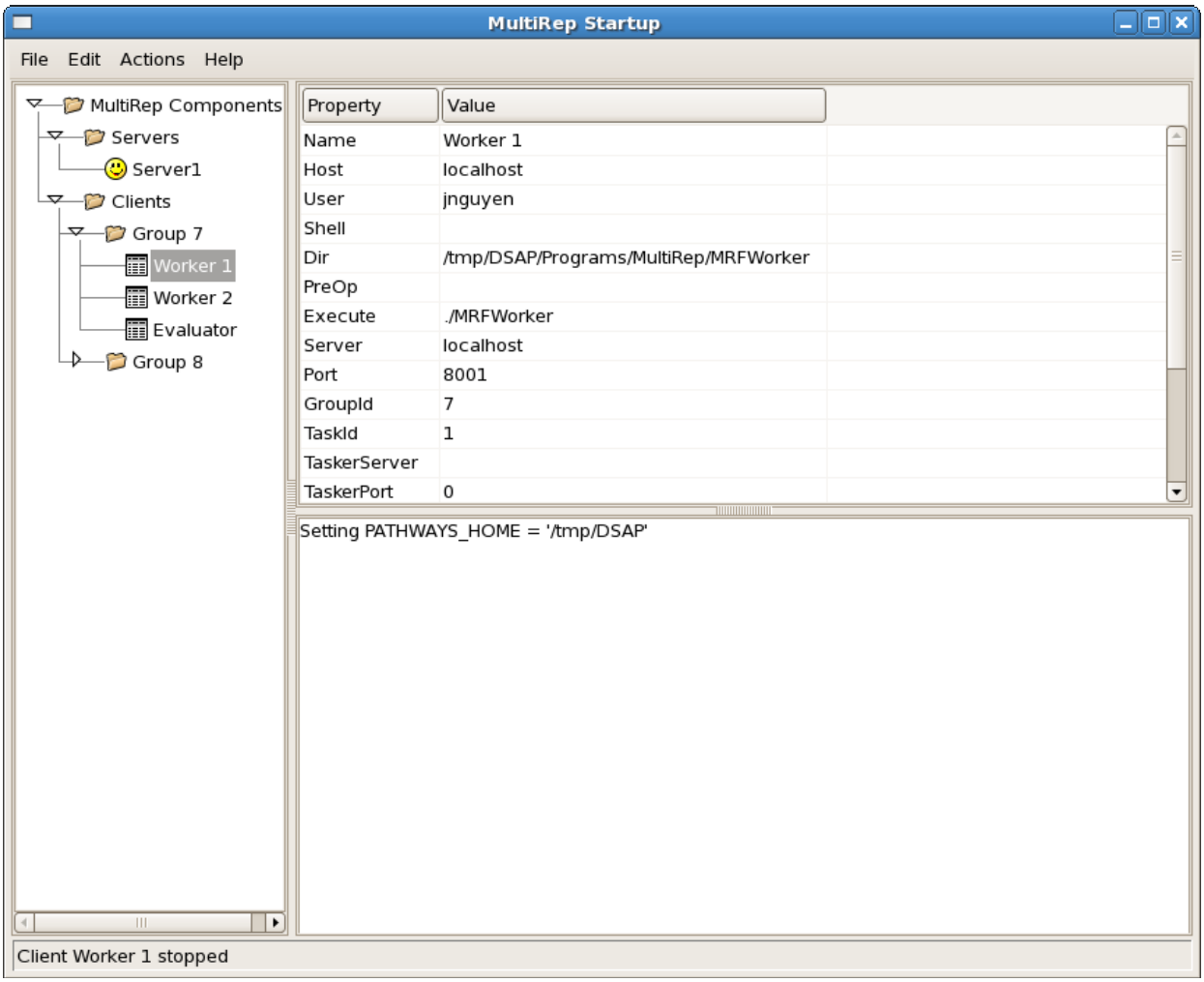

Figure 3-8: DSAP GUI front-end interface - MR_Startup GUI

3.2.4 MRFWorker

The Worker component, *MRFWorker*, receives simulation tasking from the server and launches predictive JSAF executions that run faster-than-real-time. The Worker is responsible for launching JSAF replications faster-than-real-time for a predetermined length of time. The Worker receives a command from the *MRF_Manager* to launch a JSAF execution. This command is accompanied by parameter sets (specifying the *SimRate*, start time, end time and other variables), environment variables and scenario spreadsheets. The Worker also packs up results from the replication execution in spreadsheet format and sends the information to the *MRFEvaluator*.

Upon completion of the replication, the Worker saves the state of the simulation to disk and sends it to the *MRFEvaluator* for later evaluation and comparison with real-time data and the plan objectives. Replications that fail to meet the plan objectives can be pruned by Command Staff using the *Evaluator GUI*.

The Sequence Diagram specifying the operation of the Worker executing faster-than-real-time JSAF scenarios is shown in Figure 3-10 with respect to the rest of the MRF. This is the heart of the predictive capability for DSAP/MRF.

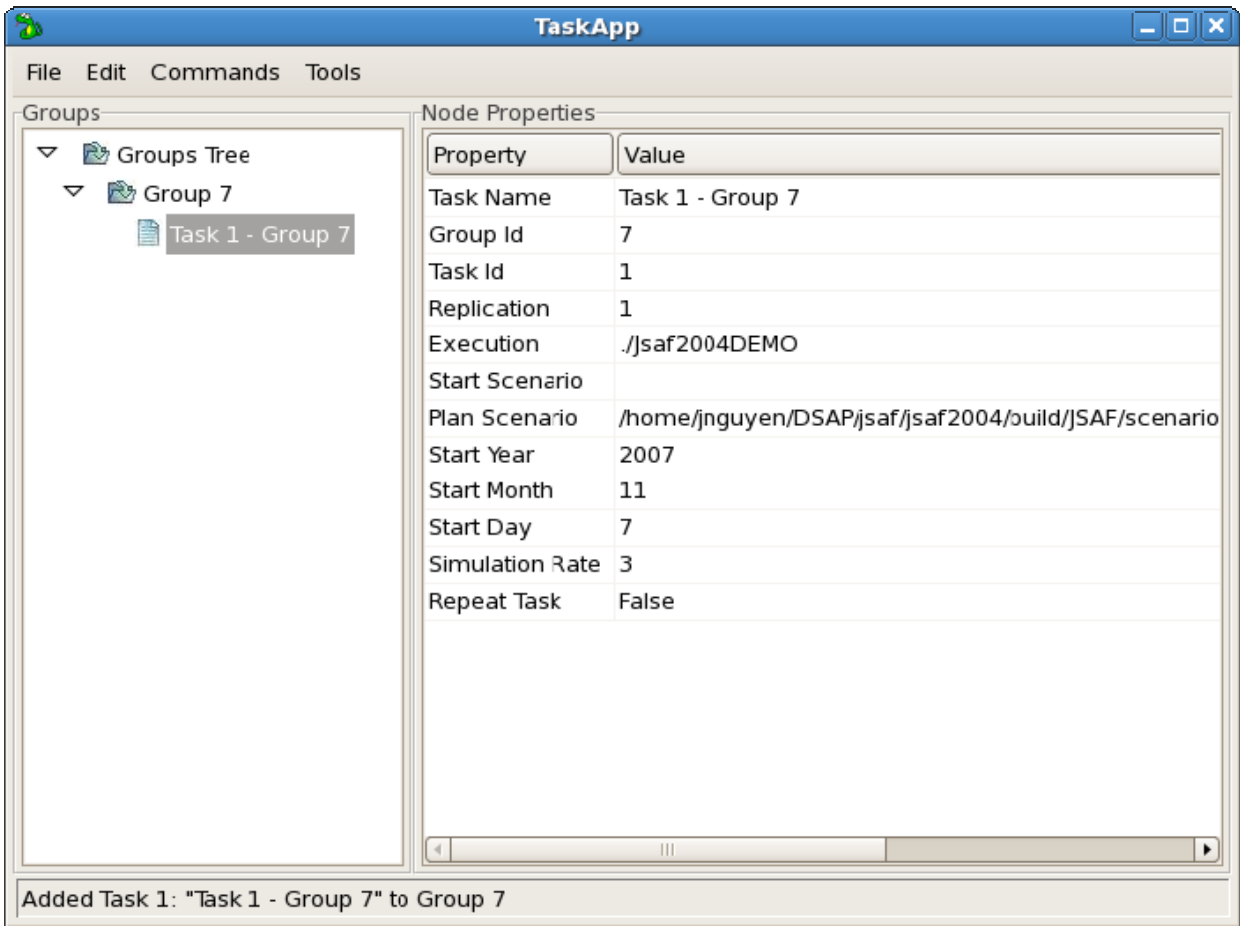

Figure 3-9: DSAP GUI front-end interface - TaskApp GUI

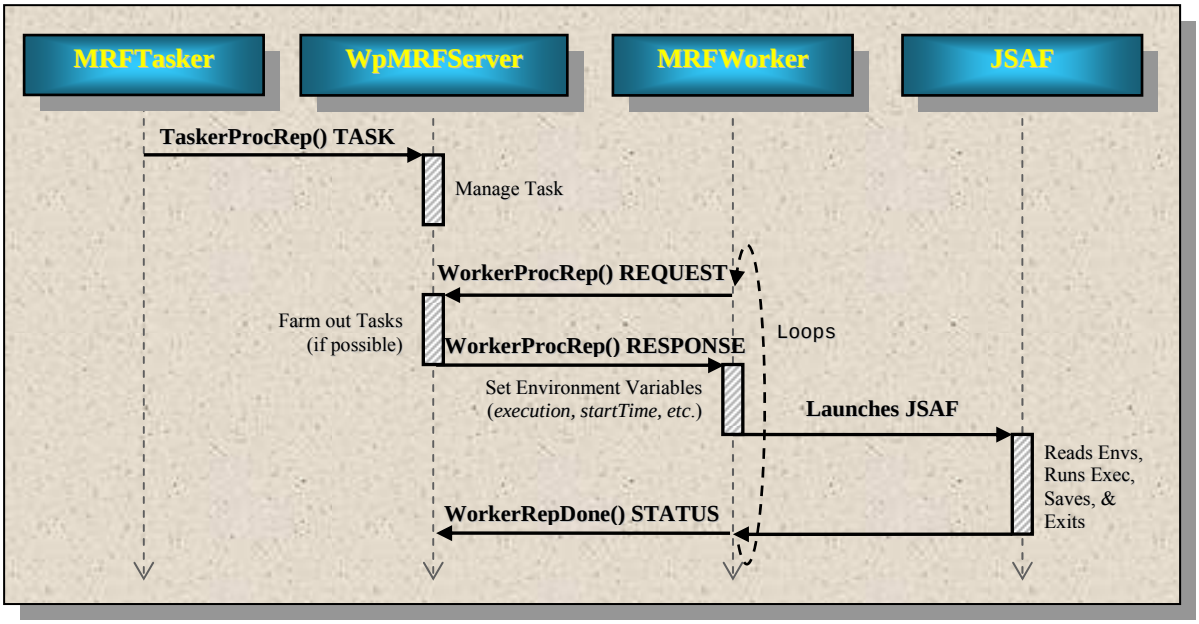

Figure 3-10: MRFWorker Control Flow for Executing Faster-Than-Real-Time Simulations

3.2.5 Plan Evaluator

The Plan Evaluator component, *MRFEvaluator*, is responsible for comparing the state of the saved faster-than-real-time replication results with the plan objectives. The *MRFEvaluator* is a Worker that evaluates the results of the JSAF replication executions; and those results can be used for evaluating against other results. The *MRFEvaluator* evaluates each of the result spreadsheets and sends them to *Evaluator GUI* for the Commander to take appropriate COAs such as determining the "best" plan or pruning plan that failed to meet the threshold. The control flow for the *MRFEvaluator* is shown in Figure 3-11, when considering the sequence of operations between the *MRFEvaluator*, *MRF_Manager*, *MRFWorkers*, and *Evaluator GUI*. The *MRFEvaluator* requests tasks from the server. When results spreadsheets are available at the *MRF_Manager*, those results are tasked to the *MRFEvaluator*, which evaluates the effectiveness of each plan. The effectiveness results are then sent to the *Evaluator GUI* for the Commander to analyze which COAs results in the "best" plan.

Workers (one or more of *MRFEvaluators*) are used to evaluate the predictive simulation executions with respect to their objectives. The evaluation process computes both the *Raw Effectiveness* and the *Relative Effectiveness* of each replication. These Measures Of Effectiveness (MOEs) are used to rank each COA and determine if each plan is consistent with the plan objectives. An alternative plan may be selected, or the existing COA may be maintained depending on its effectiveness. Also, if an alternative COA is no longer valid, that COA can be pruned or replaced by a valid alternative COA. Scenario data within the MRF's evaluation process is also updated to reflect changes in assets and resource status. At that point, the COAs and alternates are once again farmed out and executed on the available processors.

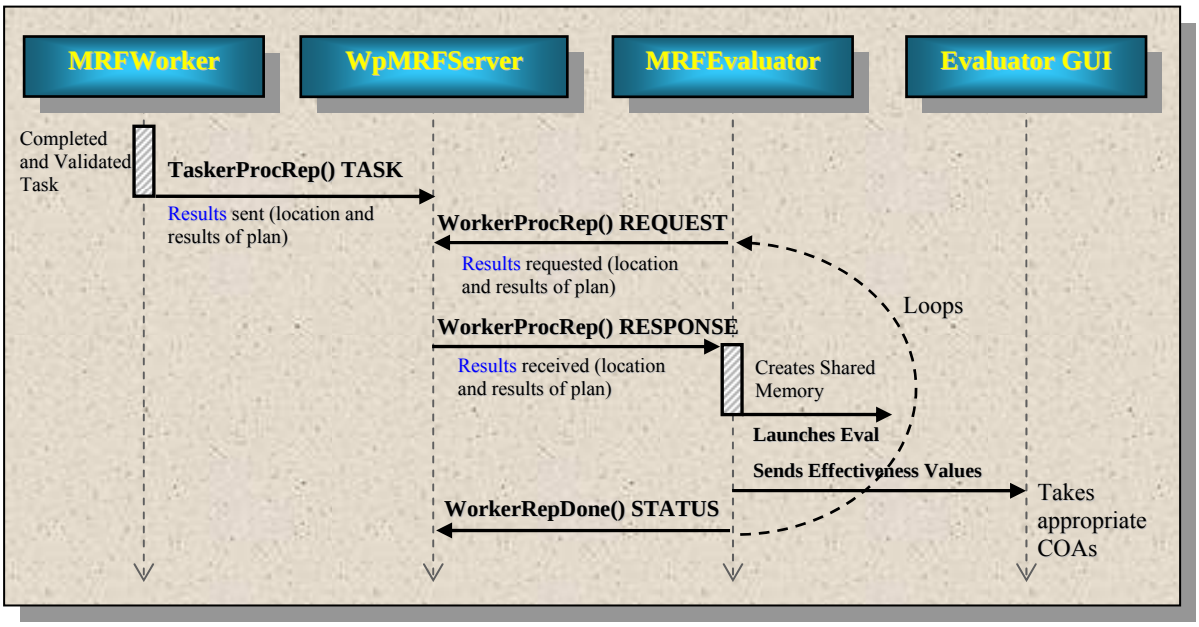

Figure 3-11: Control Flow for MRFEvaluator

The *Evaluator GUI* is shown in Figure 3-12. The GUI provides a graphical interactive interface to the MRF that allows Commanders to assess the simulation plans and replications, view the progress and performance of the plans, prune ineffective plans, and issue COAs to correct any problems. Also, the GUI provides a drill down feature, a *Details* button after selecting a line item on the Evaluation Results, to help Commanders in getting more information on the progress

of the plans or in getting more analyses of ineffective plans so that timely, effective decisions and assessments can be made to select the appropriate COA. These data or metrics can be used to gauge the effectiveness and performance of the Commander's plan.

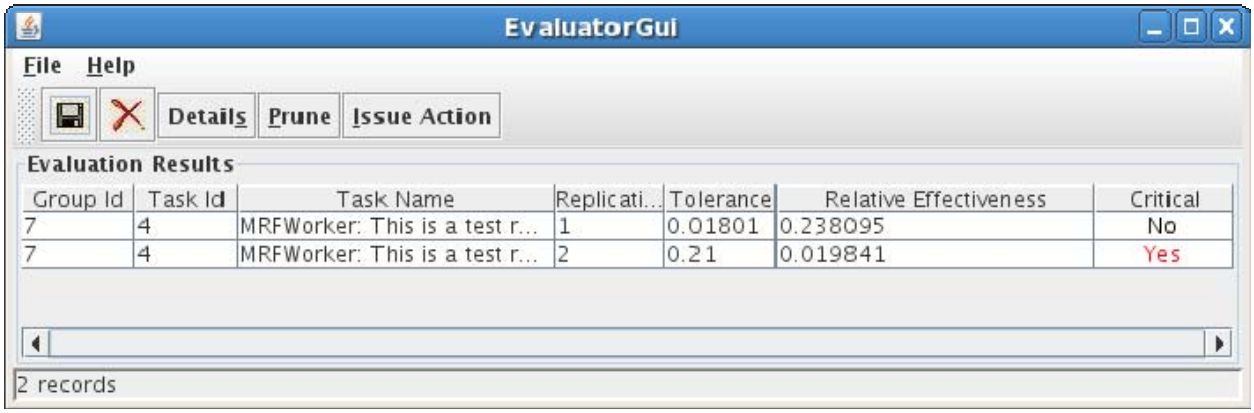

Figure 3-12: The Evaluator GUI

Details information can be viewed by the Commander as shown in Figure 3-13. When a line item on the Evaluation Results is selected and *Details* button is clicked, the detailed information can be used to create various reports such as plan assessments reports to damaged reports.

Because of the uncertain nature of predicting the outcome of plans, it is important to execute multiple replications of a plan and statistically analyze the results. The MRF calculates the mean and standard deviation of the effectiveness values for each time step in the simulation. These mean values and their standard deviations can be fitted using $\chi 2$ analysis to obtain time-based curves that provide trend analysis. The $\chi 2$ analysis is used to compare both the simulated and observed results with the expected results of the plan.

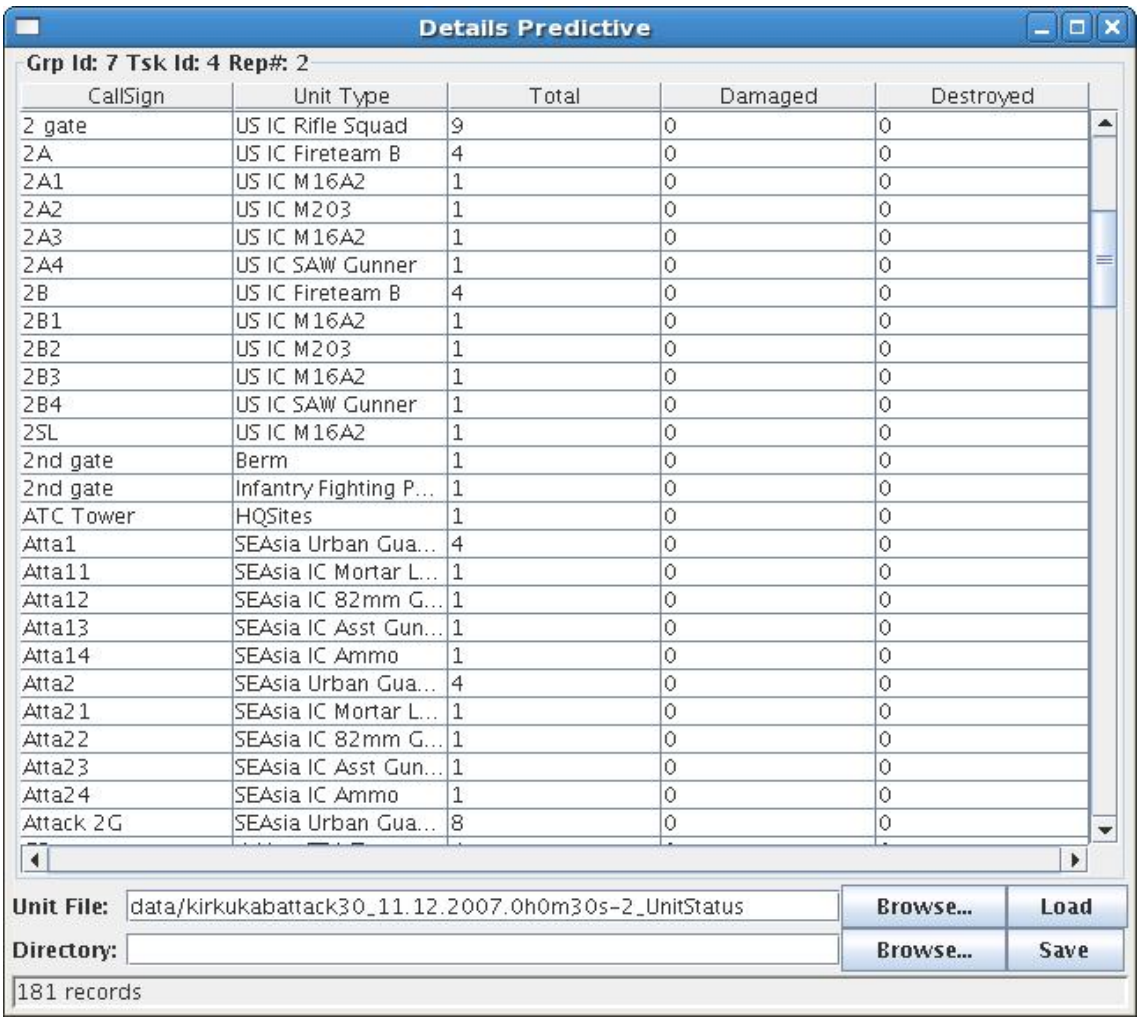

Figure 3-13: Details information on an evaluation result

4.0 DSAP MRF Modifications Supporting Dynamic Situational Awareness

The Section 3.0 focused on implementing updates to the predictive analysis components. This effort focuses on implementing the dynamic situational awareness components and the modifications that were made to the DSAP framework.

4.1 Dynamic Situational Awareness Operation

Figure 4-1 illustrates the dynamic situational awareness capability of DSAP. This basically represents scenario of the current plan being simulated in real-time. The real-time simulation, in our case JSAF, tracks the real-time operational picture, while the calibrated real-time simulation being updated with Blue and Red Force information via real-time updates or data sources. Real-time updates are provided to the real-time simulation of the current plan to correct the predicted behavior of the simulated COA. At specified checkpoints, the updated simulation can be saved to estimate the state of the real-time operational picture. Based on the decisions regarding these state-estimation metrics, with respect to plan objectives, the current plan can be pruned or a new plan can be selected to be used as a replacement.

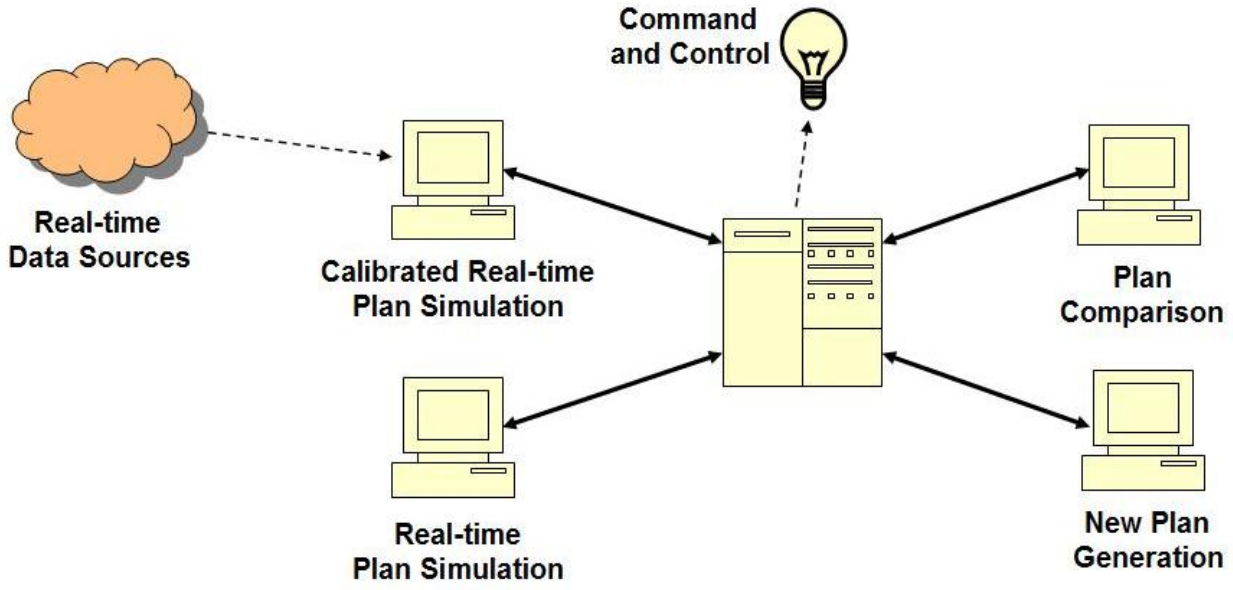

Figure 4-1: Dynamic Situational Awareness

4.2 MRF Components

For the Dynamic Situational Awareness operation, the MRF also contains three basic types of components: Taskers, Workers, and a Manager. These also follow the agent structure presented in Figure 3.3. “Action” agents are implemented as Taskers that function to task the manager with applications for workers to run, and specify how these applications should be initialized. “Work” agents are implemented as Workers that complete the tasks assigned by the manager, including executing the simulations, and saving the results of the simulations. “Evaluation” agents are implemented as Workers that evaluate and compare the results of the simulations to the plan objectives. The manager tracks time segments of simulation executions, assigns tasks to

the workers, and tackles flow control issues. The connectivity of each of these components for the Dynamic Situational Awareness operation is the same as shown in Figure 2-6. The role of each of the specific MRF components for this effort is discussed in the following subsections.

4.2.1 MRFRtTasker

The *MRFRtTasker* component issues a task to the server to initiate the real-time worker. The *MRFRtTasker* provides connectivity to the server and allows the user to specify initialization parameters such as the simulation execution name, initial scenario file, start and end time, simulation rate, and replication number.

4.2.2 MRFRtWorker

The Worker component, *MRFRtWorker*, receives simulation tasking from the server and launches JSAF executions that run real-time. The Worker is responsible for launching JSAF replications in real-time for a predetermined length of time. The Worker receives a command from the *MRF_Manager* to launch a JSAF execution. This command is accompanied by parameter sets (specifying the *SimRate*, start time, end time and other variables), environment variables and scenario spreadsheets. The Worker also packs up results from the replication execution in spreadsheet format and sends the information to the *MRFRtpEvaluator*.

Upon completion of the replication, the Worker saves the state of the simulation to disk and sends it to the *MRFRtpEvaluator* for later evaluation and comparison with real-time data and calibrated data. Replications that fail to meet the plan objectives can be pruned by Command Staff using the *Evaluator GUI*.

The Sequence Diagram specifying the operation of the Worker executing real-time JSAF scenarios is shown in Figure 4-2 with respect to the rest of the MRF. This is the heart of the situational awareness capability for the DSAP/MRF.

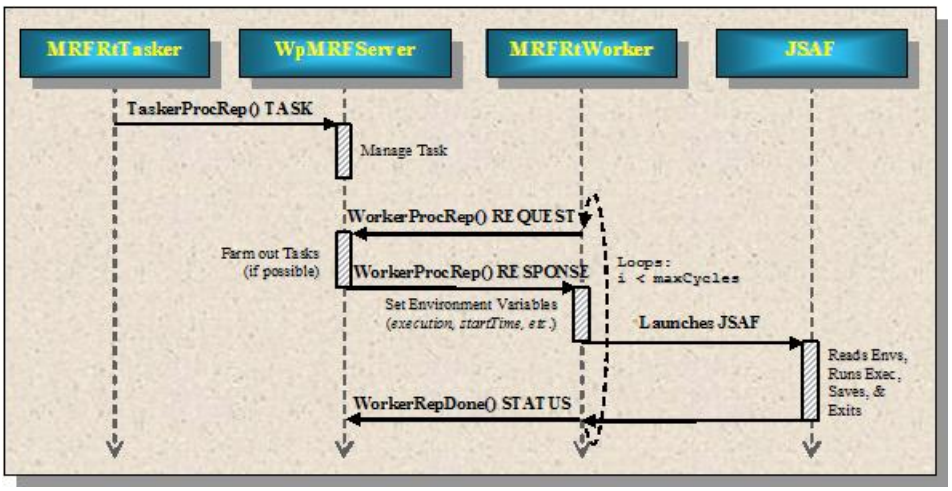

Figure 4-2: MRFRtWorker Control Flow for Executing Real-Time Simulations

4.2.3 MRFCalibratedWorker

The Worker component, *MRFCalibratedWorker*, receives simulation tasking from the server and launches JSAF executions that run in real-time. The Worker is responsible for launching JSAF replications in real-time for a predetermined length of time. The Worker performs calibration

that provides real-time updates to a persistent object (PO) that updates locations, creates a new PO entity, or deletes a PO entity, just to name a few. The Worker receives a command from the *MRF_Manager* to launch a JSAF execution. This command is accompanied by parameter sets (specifying the *SimRate*, start time, end time and other variables), environment variables and scenario spreadsheets. The Worker also packs up results from the replication execution in spreadsheet format and sends the information to the *MRFRtpEvaluator*.

Upon completion of the replication, the Worker saves the state of the simulation to disk and sends it to the *MRFRtpEvaluator* for later evaluation and comparison with real-time data and calibrated data. Replications that fail to meet the plan objectives can be pruned by Command Staff using the *Evaluator GUI*.

The Sequence Diagram specifying the operation of the Worker executing real-time JSAF scenarios is shown in Figure 4-3 with respect to the rest of the MRF. This is the heart of the situational awareness capability for the DSAP/MRF.

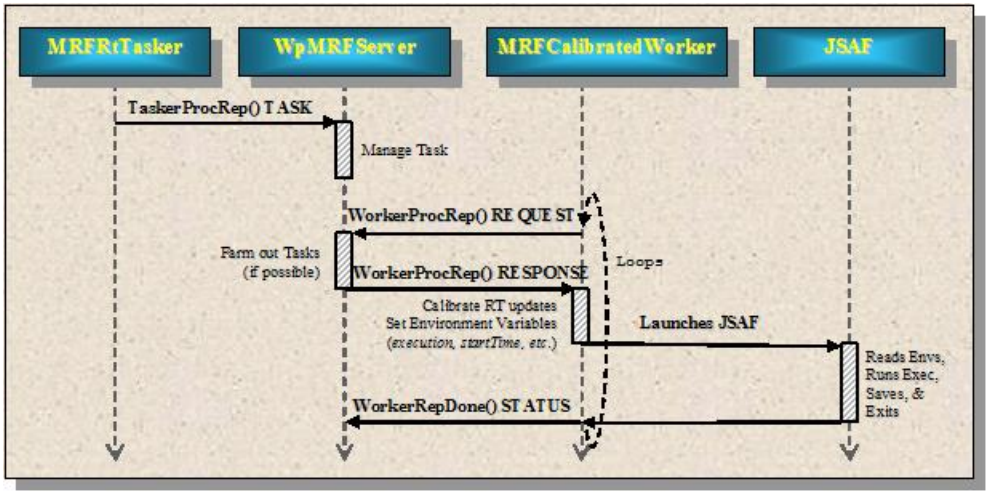

Figure 4-3: MRFCalibratedWorker Control Flow for Executing Real-Time Simulations

4.2.4 MRFRtpEvaluator

The Plan Evaluator component, *MRFRtpEvaluator*, is responsible for comparing the state of the saved real-time replications with the calibrated plan objectives. The *MRFRtpEvaluator* is a Worker that evaluates the results of the JSAF replication executions. The *MRFRtpEvaluator* evaluates both result spreadsheets of the JSAF launched by the *MRFRtWorker* and *MRFCalibratedWorker* (Figure 4-4) and sends them to *Evaluator GUI* for the Commander to take appropriate action. The control flow for the *MRFRtpEvaluator* is shown in Figure 4-5, and depicts the sequence of operations between the *MRFRtpEvaluator*, *MRF_Manager*, *MRFRtWorker*, *MRFCalibratedWorker* and *Evaluator GUI*. The *MRFRtpEvaluator* requests tasks from the server. When results spreadsheets are available at the *MRF_Manager*, those results are tasked to the *MRFRtpEvaluator*, which evaluates the effectiveness of each plan. The effectiveness results are then sent to the *Evaluator GUI* where the Commander can decide if the plan is meeting its objectives.

Worker *(MRFRtpEvaluator)* is used to evaluate the situational awareness simulation executions with respect to their objectives. The evaluation process computes both the *Raw Effectiveness* and the *Relative Effectiveness* of each simulation. These MOEs are used to determine the

effectiveness of the plan being simulated. An alternative plan may be selected or the existing COA may be maintained depending on its effectiveness. Scenario data within the MRF's evaluation process is also updated to reflect changes in assets and resource status.

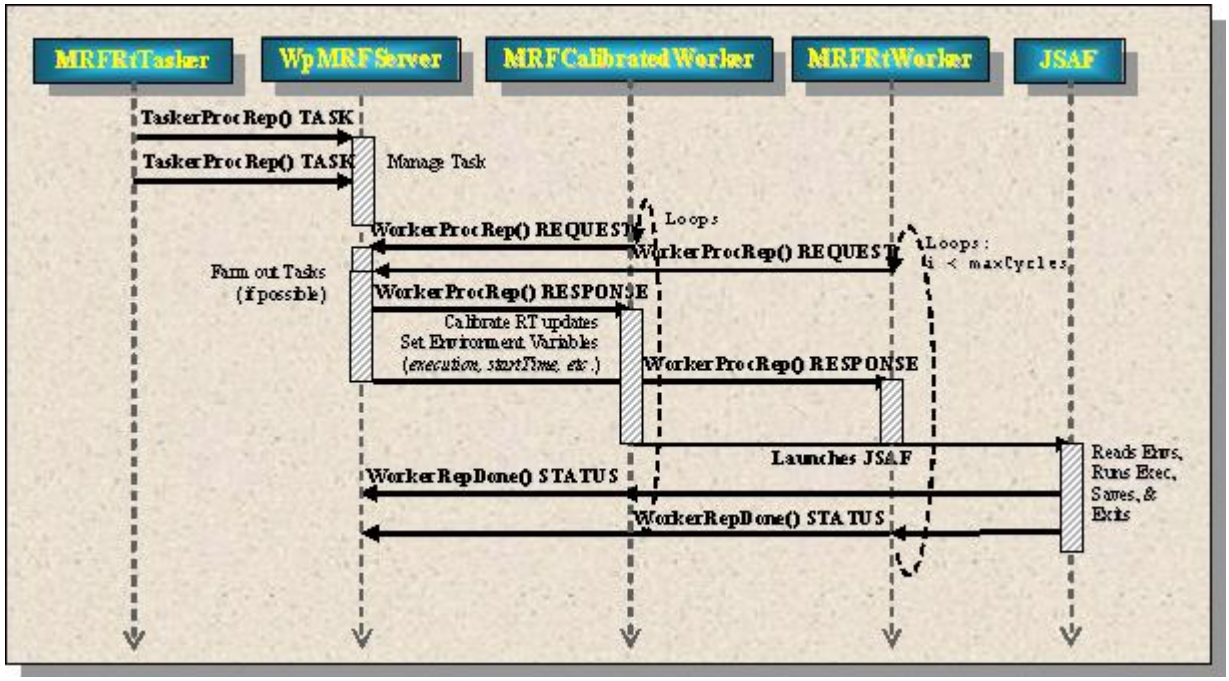

Figure 4-4: Control Flow of MRFCalibratedWorker and MRFRtWorker

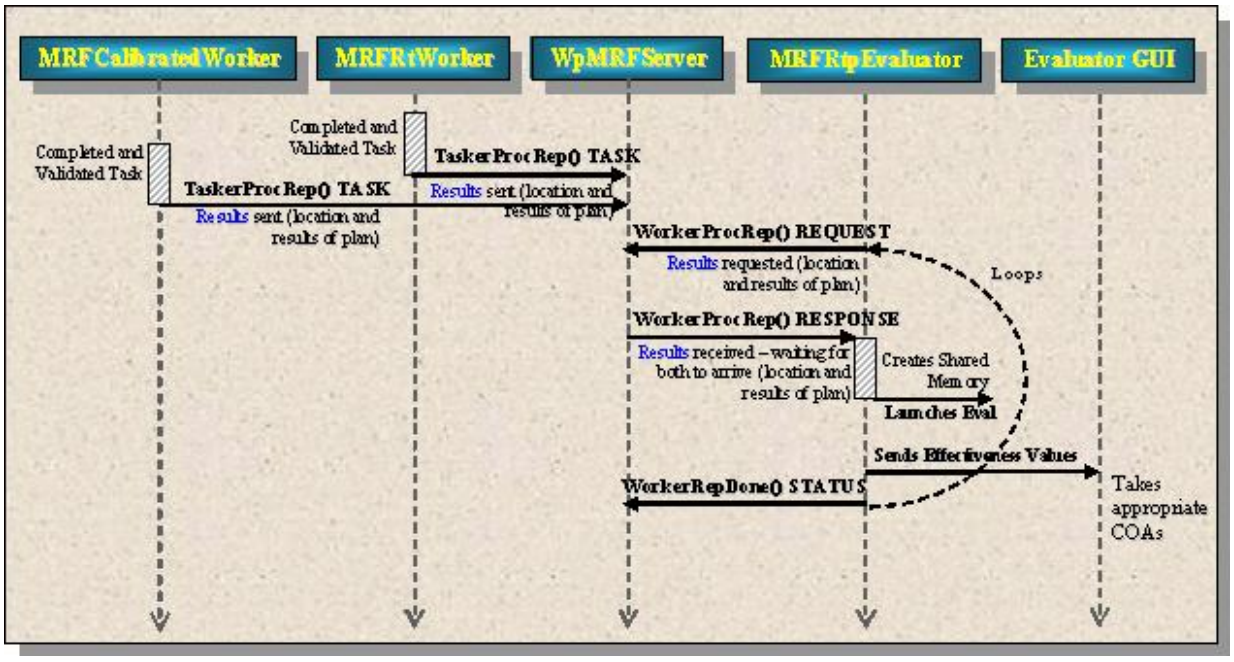

Figure 4-5: Control Flow of MRFRtpEvaluator

4.3 Modification of Objects in MRF/JSAF

Functions were implemented to allow persistent object (PO) database updates. These functions create and modify objects in JSAF using the DSAP framework. The update process utilizes the checkpoint and PO entry file approach. The effort is described in the following subsection.

4.3.1 Checkpoint/PO Entry File Approach

This update process allows the modification of simulation state in JSAF. JSAF (or other simulations) simulations can be checkpointed at a user-defined point in time. The checkpoint data is used as the starting point to initialize and bring JSAF to its previous state before it can be updated or calibrated with real-time information defined in a PO entry file. A script or application uses data source information of real-time simulations to generate the PO entry file.

The PO entry file is a comma delimited text file. Currently, the PO entry file supports a PO entry location update and a PO entry deletion. The PO entry location update format takes a form of a "UPDATE_LATLON" tag, callsign identification, and location (in latitude/longitude, e.g. N##/##/##.###,E##/##/##.###). The PO entry deletion format takes a form of a "DELETE" tag and callsign identification.

This approach requires a timely checkpoint and startup process. This approach requires making specific JSAF API calls. Also, this approach allows specific updates of information that are relevant to the current simulation. Other PO entries not updated are left intact. The code added to JSAF source code is shown in the snippet box below.

```
void ProcessUpdates(char *updatesFile, PO_DATABASE *po_db) {
   FILE *rfd;
   char dataBuffer[256];
   string strData;
   int start, end, count, newpos;
   string strSupportCode;
   string sub_string;
   string callsign;
   string s_lat;
   string s_lon;

   cout << "ProcessUpdates from file: " << updatesFile << endl;

   rfd = fopen(updatesFile, "r");

   if (rfd != (FILE *) NULL) {
      while (fgets(dataBuffer, 256, rfd) != NULL) {
         start = 0;
         end = 0;
         count = 0;
         newpos = 0;

         strData = dataBuffer;

         // Trimming newline character
         newpos = strData.rfind('\n', strData.size());
         if (newpos != (int) string::npos) {
            strData.erase(newpos);
         }
         end = strData.find(",", start);

         if (end != (int) string::npos) {
            strSupportCode = strData.substr(start, end - start);
            start = end + 1;
            count++;

            if (strSupportCode == "UPDATE_LATLON") {
```

```
               while ((end = strData.find(",", start)) != (int) string::npos) {
                  sub_string = strData.substr(start, end - start);

                  if (count == 1) {
                     callsign = sub_string;
                  } else if (count == 2) {
                     s_lat = sub_string;
                  }

                  start = end + 1;
                  count++;
               }
               s_lon = strData.substr(start, strData.size() - start);

               cout << "Callsign = " << callsign << endl
                    << "Lat      = " << s_lat    << endl
                    << "Lon      = " << s_lon    << endl;

               float64 lat = (float64) LatLonToDouble(s_lat);
               float64 lon = (float64) LatLonToDouble(s_lon);
               float64 z   = (float64) 360;

               po_pause_me(po_db);
               UpdateUnitLocation(callsign, lat, lon, z, po_db);
               cout << "Updated: " << callsign << ": " << lat << "," << lon << endl;
               po_unpause_me(po_db);
            } else if (strSupportCode == "DELETE") {
               while ((end = strData.find(",", start)) != (int) string::npos) {
                  start = end + 1;
                  count++;
               }
               callsign = strData.substr(start, strData.size() - start);

               cout << "Callsign = " << callsign << endl;

               po_pause_me(po_db);
               DeleteUnit(callsign, po_db);
               cout << "Deleted: " << callsign << endl;
               po_unpause_me(po_db);
            } else {
               cout << "Warning: Not supported code [" << strSupportCode << "] for entry PO
update." << endl;
               }
         }
      }

      fclose(rfd);
   }
}
```

Code Segment 4-1: Code for Updating PO Entries in JSAF Simulations

4.4 Updated Evaluation GUIs

In addition to the above updates to DSAP's Dynamic Situational Awareness machinery, this effort also provided updates to the DSAP Graphical User Interfaces. For Dynamic Situational Awareness, the Evaluator GUI is shown in Figure 4-6. This version provides evaluation capabilities that display Calibrated Worker results. These calibrated results are compared against an idealized plan to see if some threshold as been passed or exceeded. This threshold is represented by a user-defined cost function.

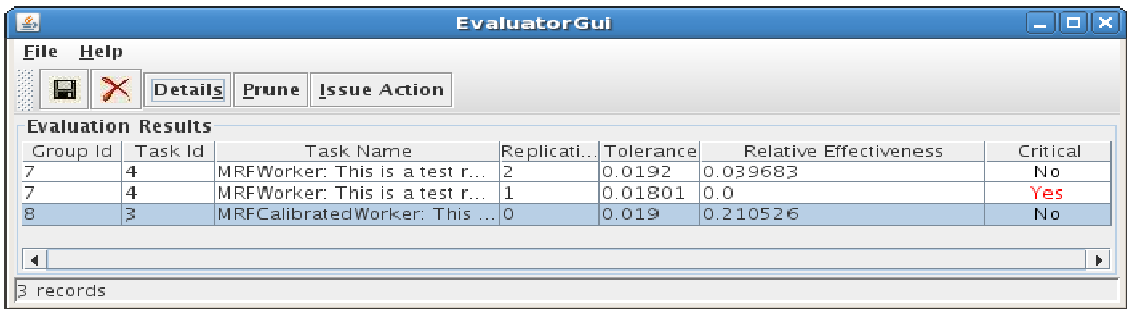

Figure 4-6: Evaluator GUI for Dynamic Situational Awareness

Additional GUI capabilities can be found when examining the *Details* tab for the Evaluator GUI. Clicking on this *Details* tab presents the GUI view shown in Figure 4-7. This graphic represents the different entities taking part in the idealized real-time plan, in comparison to the calibrated real-time plan. Entities (denoted by call sign) that are not present in both plans are highlighted in Cyan. This triggers an alert that keys the generation of new plans (by a 3rd party tool). The magenta highlights depict differences in position for targets/entities between the plans (ideal and calibrated). These differences can trigger an alert if their difference in position exceeds some threshold.

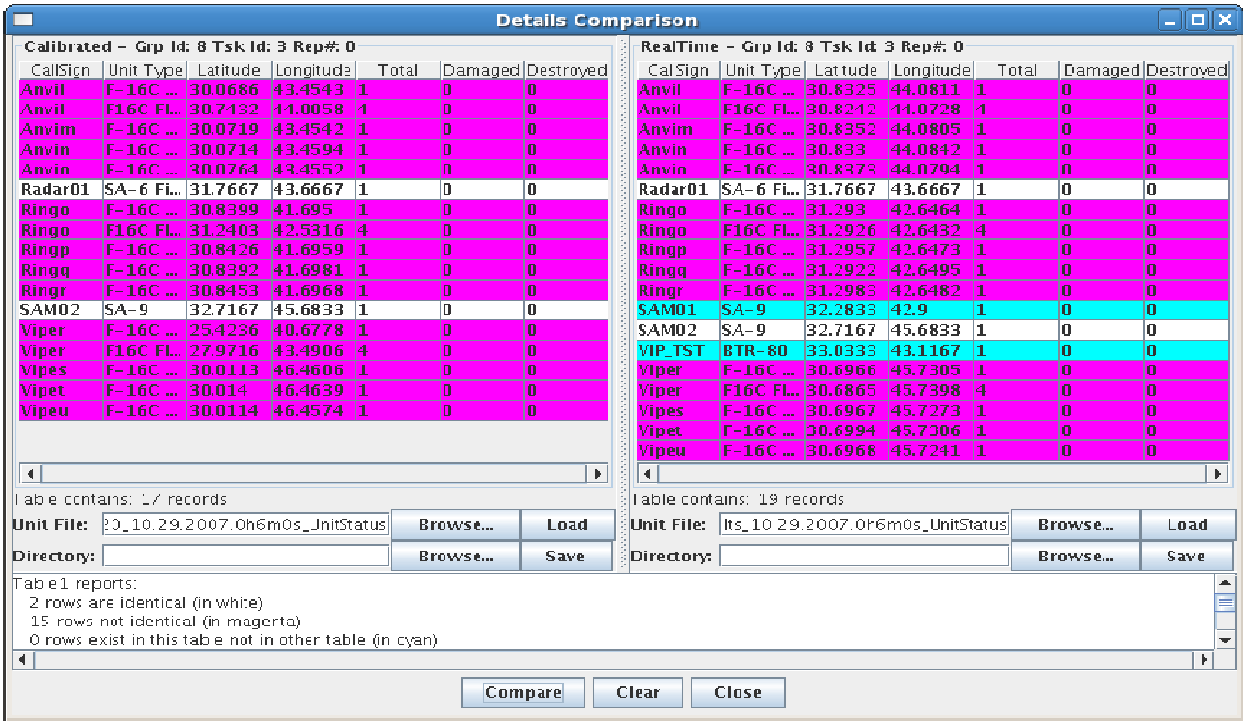

Figure 4-7: Details for Evaluator GUI with Highlights

4.5 Summary

This section has detailed the modifications and enhancements performed on this effort to DSAP's MRF to support the real-time Dynamic Situational Awareness capability for operational C2. This effort developed updates to real-time simulation components for the *MRFCalibratedWorker*, *MRFRtWorker*, and *MRFRtpEvaluator* to support idealized and calibrated simulations, re-defined the control flow for real-time dynamic situational awareness applications, and implemented mechanisms to calibrate real-time simulations from data source inputs.

5.0 Integrating with 3rd Party Tools

Several versions of the DSAP Framework have been implemented and installed on systems at AFRL's Information Systems Research Branch (IFSB) Command and Control Technology Center (C2TC). The tasks performed under this effort allow DSAP to execute plans and alternatives to evaluate "what-if" scenarios, while examining how closely calibrated plans followed their idealized paths. As part of this effort, RAM Laboratories investigated the process for integrating DSAP's Dynamic Situational Awareness mode, and DSAP's Predictive mode with 3rd party plan generation tools. Specifically, our efforts investigated the ability for DSAP to interoperate with Charles River Analytics (CRA's) STOMP tool. Figure 5-1 depicts an architecture that was defined in conjunction with AFRL IFSB personnel, AFRL IFSB contractor personnel, and CRA. The overall control flow for this architecture is as follows:

(1) A plan of the day is selected and executed using DSAP's Dynamic Situational Awareness capability. Simulations of this "plan-of-the-day" use JSAF for both the real-time simulation component and the real-time-calibrated simulation component

(2) AODB and MIDB updates are provided from TBMCS and used to calibrate the real-time calibrated simulation running on JSAF. It should be noted that the TBMCS connection was emulated using a flat file (a connection to TBMCS was unavailable) and calibration mechanisms involved altering positions of entities and/or adding targets (some high-value in nature).

(3) DSAP's MultiRepRTPEvaluator was used to generate a threshold comparison that compared cost functions against one another in support of this thresholding capability

(4) When the threshold was reached, additional plans would be generated by STOMP based on the current state of the real-time-calibrated simulation when the threshold was reached.

(5) When plan generation is completed, DSAP's Predictive mode is then used evaluate all alternatives using a user-defined cost function (defined by parameter files).

The design and implementation of this architecture would allow is to investigate DSAP's use in addressing several objectives, namely

(1) Use of DSAP's Dynamic Situational Awareness and Predictive modes of operation within a single control flow

(2) Demonstrating thresholding and comparison on Dynamic Situational Awareness side (discussed in the previous chapter).

(3) Interoperating with a 3rd party tool, using both Dynamic Situational Awareness and Predictive modes. This includes passing data between both modes of operation, and

(4) Providing a solution to interoperate in a heterogeneous computing environment (DSAP is currently executed on Linux platforms while CRA's STOMP is a Windows environment).

The following discussion covers some of the lessons learned during this effort with regard to a design for a DSAP-STOMP integrated system, as well as some of the issues that will have to be

addressed in the future. Full integration of DSAP-STOMP was not completed during this effort due to time limitations.

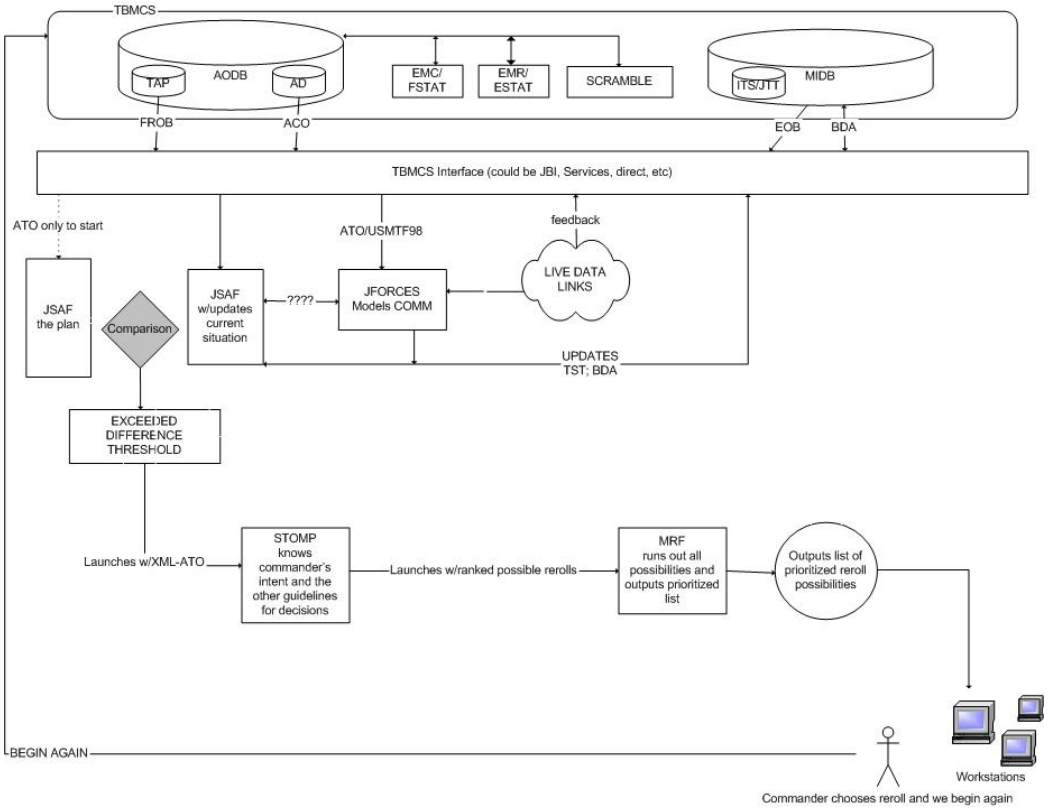

Figure 5-1: Control Flow for Integrated DSAP-STOMP

5.1 DSAP-STOMP Interoperability

The first issue discussed on this effort involved ensuring that DSAP and STOMP could pass information (primarily simulation state and plan information) between various tools. At the start of this effort, DSAP primarily used binary checkpoints for storing and restarting from operational state data. These checkpoints had been used since the framework evolved to support JSAF2004. (Initial versions used JSAF5.33b and also had a spreadsheet-based checkpoint restart capability). When interoperability between DSAP and STOMP was first examined, it was decided that the most feasibility approach to passing simulation state and plan information was through comma-delimited spreadsheets. STOMP research had included an effort to export their plans in comma-delimited format. DSAP had once primarily used comma-delimited formats. Based on this decision, the spreadsheet-based information store was re-inserted into the DSAP implementation (although different JSAF2004 functions were used than had previously been used under the JSAF5 implementation). Re-implementing the spreadsheet capability allowed DSAP's Dynamic Situational Awareness mode to save simulation state in comma-delimited format when a evaluated threshold was exceeded. STOMP could then use this state as a starting point for its plan development. In addition, STOMP outputted comma-delimited spreadsheets for its plan and scenario information. These spreadsheets could be read by DSAP's Predictive mode to simulate the "what-if" scenarios involving plans and their alternative.

5.2 Messaging Between DSAP-STOMP

A second issue that arose in a design for an integrated DSAP-STOMP involved the use of the proper message passing mechanisms to raise notifications (of a threshold being reached), pass simulation state, and notify DSAP that STOMP had completed a set of plans and alternatives for examination. One issue with regard to this messaging or notification process involved the fact that DSAP is a Linux-based system and STOMP is a Windows based system. To support such interoperability, it was decided that a web-service message be used for communication. DSAP could send a notification as a web service when a threshold had been reached (telling STOMP to start generating new plans). Likewise, STOMP could use a web service to tell DSAP that plans had been generated in a directory (telling DSAP to start its predictive evaluation of those plans). One issue that arose was passing simulation state (i.e. scenario) information from DSAP's Dynamic Situational Awareness mode to STOMP. DSAP's comma-delimited checkpoints in this area have many vacant or null fields in its comma-delimited format. STOMP requires these fields to be populated for its plan generation process to operate properly. Future efforts that integrate DSAP and STOMP will have to designate responsibility for providing this information.

In addition to the above issues, STOMP will have to use a web service to notify DSAP that it has generated a set of plans for analysis in a given directory. As such, the notification must specify the location of this directory, or the plan information must be stored in a database or directory in a pre-determined location (which may be tricky in that it must be accessed by both the Windows based STOMP and the Linux-based DSAP). A potential alternative to this solution would be to pass all plans and scenario information as a XML-based data model. Subsequent versions of JSAF, such as JSAF2007, have better capabilities in this area and should be examined to investigate their ability to support such a feature.

6.0 Summary and Future Work

The DSAP Framework is being used to provide a prototype Predictive Analysis and Dynamic Situational Awareness capabilities for plans derived from Air Tasking Orders (ATOs) and their alternatives while calibrating those plans with real-time C4I database inputs. The DSAP Framework utilizes JSAF as both its predictive (faster-than-real-time) and real-time simulation components, and pulls C4I information from Theater Battle Management Core System's (TBMCS) Air Operations Database (AODB) and Modernized Integrated Database (MIDB) as well as a variety of flat files.

This effort streamlined the control flow for DSAP's MRF in order to ensure that both predictive and dynamic situational awareness modes can simultaneously operate on a single platform. In addition, this effort re-architected the MRF into a Work-Action-Evaluate paradigm (based on RAM Laboratories' Extensible Grid) to ensure that additional components could plug-and-play with DSAP. Such components could be plan generation tools, such as STOMP, plan optimization tools, evaluation capabilities, and additional simulations and models. Much of the work was spent moving functionality out of the old DSAP Manager (Server) component and into various Workers and Taskers. The resulting environment is one that can much more easily support 3rd party tools, simulations and components, providing an improved DSAP that can be more readily used in Advanced Concept Experiments or Exercise.

In addition to the above development, the DSAP Framework was installed at AFRL by RAM Laboratories in May and September of 2007. Additional updates and enhancements were also provided as patches or clean installs that were performed by AFRL personnel, or AFRL contractor personnel. This installation and demonstration process resulting in the development of a variety of tutorials, installation guides, and user documentation to guide the installation and use process for DSAP. Also, work was spent on developing "installer" capabilities and additional GUIs to facilitate DSAP use.

To continue to build on successes of our DSAP work, future efforts will focus on using DSAP with additional models and simulation environments, calibrating with additional data sources, and support additional evaluation capabilities to include custom Graphical User Interfaces and/or 3rd party COTS/GOTS evaluation and analysis tools. Selections of potential tasks are discussed in the following paragraphs.

6.1 Further the Effort to Calibrate with Real-Time Data

Current and past DSAP implementations have demonstrated that DSAP can be connected to and calibrated with data from flat files, spreadsheets, and databases such as Oracle and Sybase. In particular, DSAP has connected to Oracle and Sybase databases such as AODB and MIDB tables from an unclassified version of TBMCS. Future efforts will work to implement techniques to extract data from additional sources including the TMDB and additional data sources to include the following.

6.1.1 Connect to TMDB

Future efforts can connect the DSAP Framework to the Track Management Database (TMDB) for the purpose of extracting real-time track data and using that data to calibrate real-time

simulations. Subsequent efforts can focus on implementing mechanisms to calibrate DSAP simulations with TMDB data.

6.1.2 Connect to XML-based Data Sources

Future efforts will work to connect to XML-based data sources. These data sources can be plans generated by 3rd party plan generation tools, or data models such as the Joint Consultation Command and Control Information Exchange Data Model (JC3IEDM) to exchange data with regard to operational state or plan/Course of Action information.

6.2 Improved Installation of DSAP and the MRF

Installation of DSAP requires certain libraries and compilers for Linux platforms. At the tail end of this past effort, RAM Laboratories developed a DSAP installer that installs DSAP and all its support libraries (including the Xerces library that was particularly troublesome). Future efforts will further elaborate upon the install process to ensure DSAP can be installed on platforms supporting Linux and other flavors of Unix.

6.3 Web Service implementation

This effort entailed working with AFRL personnel and 3rd party contractors (Charles River Analytics) to define the process for integrating and interoperating DSAP with other 3rd party tools such as CRA's STOMP tool. These additional tools may be developed on other operating systems and utilize other data formats and models for capturing their information. Discussions on this effort resulted in the decision to use a web service to communicate notifications between DSAP and 3rd party tools that may use that information. Future efforts will elaborate on this approach by implementing web services in Java, describing the service using Web Service Description Language, using XML-based data models (such as C-BML or the JC3IEDM), and taking advantage of existing registries. As such DSAP can be made into a useful tool (service) on the Global Information Grid that provides simulation and evaluation services, while taking advantage of simulation, evaluation, analysis, and plan generation services in additional to web-service based C4I data sources that can be discovered.

6.4 Graphical User Interface Development

This effort developed GUIs that guide the user in starting, configuring, and managing DSAP tasks and their corresponding parameters. Additional GUIs were constructed that allowed the user to visualize evaluation results and "drill down" to "Details" with regard to simulation state. Real-time evaluation capabilities graphically displayed differences between real-time (idealized) entities and calibrated entity values. Additional work in GUI development is two-fold: (1) supporting a capability that allows users to select parameters and entity information for consideration (which will be reflected back to the running simulations in a manner that will allow those simulations to capture and save the desired state information), and (2) supporting a web-based capability that interfaces with Evaluate components (built on Extensible Grid Workers) that either allows 3rd party analysis users to utilize DSAP simulation capability as a service, or allows DSAP users to utilize 3rd party services in its evaluation process.

6.5 Research and Development into Integrating and Interoperating with 3rd Party Mechanisms

This effort held discussions with AFRL personnel and contractor personnel with regard to integrating and interoperating DSAP with 3rd party planning tools (i.e. CRA's STOMP). The effort resulted in DSAP being used to read and install STOMP-generated plans from a directory and execute those plans on available Workers using JSAF as the simulation tool. Discussions on integrating and interoperating DSAP with STOMP or other 3rd party tools also entailed using a web service for notification of certain evaluation thresholds being met (for instance, when the calibrated simulation differs beyond some defined point with respect to the idealized simulation/plan, such when a previously not-considered high value target appears). Additional work will entail implementing this web service, in addition to other fields. Based on our discussions, these fields most likely represented XML-based or comma delimited scenario state information based on the plan being executed. Specific subtasks that will have to be addressed are covered in the following paragraphs.

6.5.1 Interface Definition

Future work will involve working with government personnel and selected tool vendors to define a common interface and API that will enable 3rd parties to work with DSAP. This interface should support the web-service paradigm and will seek to use and support standard data models and tools.

6.5.2 Interface Development and Integration

Future work will involve developing mechanisms such as web services that will allow 3rd parties to use DSAP's simulation and evaluation capabilities. In addition, interfaces will be developed that will allow DSAP users to leverage existing 3rd party tools and application on the evaluation and analysis. An obvious candidate for developing interfaces in this area would be CRA's STOMP tool. Developing such an interface will not only demonstrate DSAP's use in supporting a web-oriented paradigm, but will also enhance DSAP capabilities by pulling in plan generation mechanisms.

6.6 Exercise and Experiment Support

Future DSAP efforts will involve participating in exercises and experiments such as the Advanced Concept Experiment (ACE). Support for these efforts will entail ensuring that DSAP can be easily installed on platforms at the Experiment/Exercise site, exercise/experiment participants can be readily trained on using DSAP, techniques or mechanisms are in place to capture metrics from the experiment/exercise, and an advanced DSAP implementation is developed that takes into account the scenarios/plans being executed, the simulations being used, the data sources being used for calibration, the desired simulation state objectives, and the evaluation and analysis mechanisms that are desired for the exercise and/or experiment.

7.0 Bibliography

John R. Surdu. Connecting Simulation to the Mission Operational Environment. Ph.D. Thesis. Texas A&M University. 2000

Alex F. Sisti. “Dynamic Situation Assessment and Prediction (DSAP)” Proceedings of SPIE, Enabling Technologies for Simulation Science VII Vol.5091. 2003.

Dr. Paul Phister, Dr. Timothy Busch, and Igor Plonisch. “Joint Synthetic Battlespace: Cornerstone for Predictive Battlespace Awareness.”

Reaper Jerome, Trevisani Dawn, and Alex Sisti. “Real-Time Decision Support System (RTDSS)” Proceedings of the Western MultiConference. January, 2003.

McGraw Robert, Lammers Craig, and Steinman Jeff, 2004. “Software Framework in Support of Dynamic Situation Assessment and Predictive Capabilities for JSB-RD”. In proceedings of the SPIE - Enabling Technologies for Simulation Science VIII Conference.

McGraw Robert, Lammers Craig, and Trevisani Dawn, 2004. “Dynamic Situation Assessment and Predictive Capabilities in Support of Operations”. In proceedings of the Fall Simulation Interoperability Workshop, Orlando, FL. 2004.

McGraw Robert, Lammers Craig, Steinman Jeff, and Trevisani Dawn, 2005. “A DSAP Framework for the Global Information Grid's Modeling and Simulation Community of Interest”. In proceedings of the *Spring Simulation Interoperability Workshop*, San Diego, CA. 2005.

Lammers Craig, McGraw Robert, and Trevisani Dawn, 2005. “Applying a Multireplication Framework to Support Dynamic Situation Assessment and Predictive Capabilities”. In proceedings of the SPIE - Enabling Technologies for Simulation Science IX, Orlando, FL. 2005.

Effects Based Operations. Available: http://www.afrlhorizons.com/Briefs/June01/IF00015.html

Theater Battle Management Core Systems (TBMCS). Available http://jitc.fhu.disa.mil/tbmcs/tbmcs.htm.

Available http://www.mstp.quantico.usmc.mil/modssm2/InfoPapers/INFOPAPER%20JSAF.htm

Numrich, S.K., Hieb, M., and Tolk, A. “M&S in the GIG environment: An Expanded View of Distributing Simulation” Presented at the Interservice Industry Training Simulation Education Conference. Orlando, FL. 2004.

http://e-mapsys.com/C2IEDM-MIP_Overview_20Nov2003.pdf

]Steinman Jeff, 2002. “The Standard Simulation Architecture.” In proceedings of the 2002 SCS Summer Computer Simulation Conference.

Douglas Schmidt. ACE+TAO. Available http://www.cs.wustl.edu/~schmidt/.

Bailey Chris, McGraw Robert, Steinman Jeff, and Wong Jennifer, 2001. "SPEEDES: A Brief Overview" In Proceedings of SPIE, Enabling Technologies for Simulation Science V, Pages 190-201.

RAM Object Request Broker Programming Guide, Version 1.2, DRAFT.

8.0 Acronyms

ACE	Adaptive Communication Environment
AODB	Air Operations Data Base
AFRL	Air Force Research Laboratory
ATO	Air Tasking Order
BML	Battle Management Language
C2IEDM	Command and Control Information Exchange Data Model
C2IS	Command and Control Information Systems
C4I	Command, Control, Communications, Computers, and Intelligence
CCSE	Common Component Simulation Engine
COA	Course Of Action
CORBA	Common Object Request Broker Architecture
DSAP	Dynamic Situation Assessment and Predictive
FSS	Force Structure Simulation
FTRT	Faster Than Real Time
GCCS	Global Command and Control System
GIG	Global Information Grid
GUI	Graphical User Interface
IFSB	Information Systems Branch
IITSEC	Interservice Industry Training Simulation Education Conference
JDBC	Java Data Base Connectivity
JDK	Java Developer's Kit
JSAF	Joint Semi-Automated Forces
JTIDS	Joint Tactical Information Delivery System
JWARS	Joint WarGaming System
MIDB	Modernized Integrated Data Base
MRF	Multiple Replication Framework
MSDL	Military Scenario Description Language
NATO	North Atlantic Treaty Organization
NCES	Network Center Enterprise Services
NCOW	Network Centric Operations and Warfare
ODBC	Open Data Base Connectivity

ORB	Object Request Broker
OS	Operating System
POC	Point of Contact
RT	Real Time
RTP	Real Time Picture
SATCOM	Satellite Communications
SBIR	Small Business Innovative Research
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete Event Simulation
TAO	The ACE ORB
TBMCS	Theater Battle Management Core System
XML	Extensible Markup Language